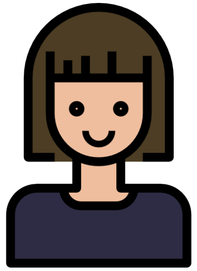
LATKE: A Framework for Constructing Identity-Binding PAKEs

Jonathan Katz
University of Maryland, Google

Michael Rosenberg, PhD(!)
University of Maryland

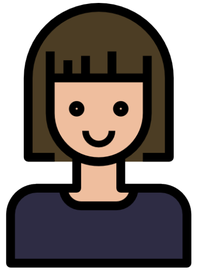
ia.cr/2023/324

What is a PAKE



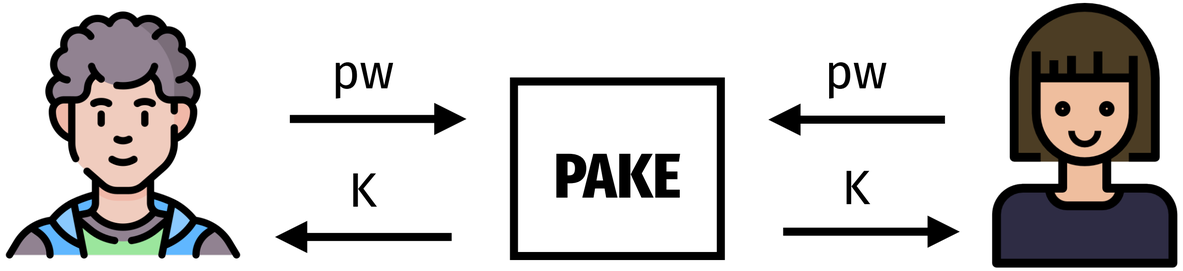
What is a PAKE

Two parties use a password to establish a secure shared secret



What is a PAKE

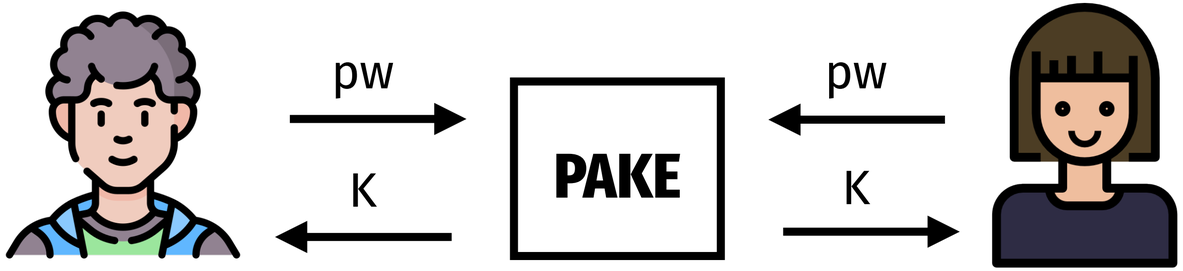
Two parties use a password to establish a secure shared secret



What is a PAKE

Two parties use a password to establish a secure shared secret

A passive adversary cannot derive K

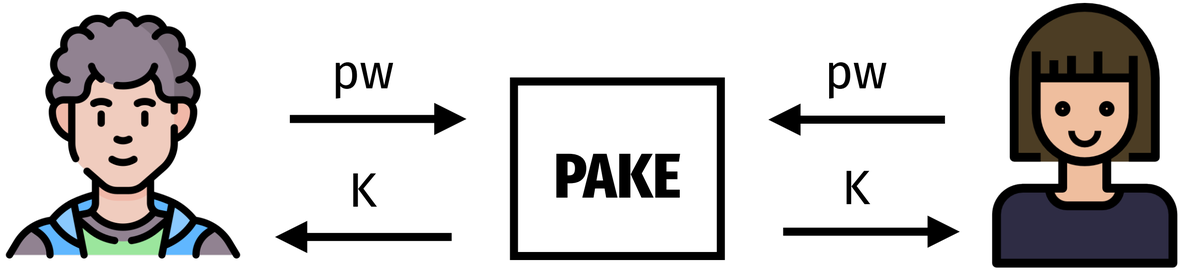


What is a PAKE

Two parties use a password to establish a secure shared secret

A passive adversary cannot derive K

An active adversary has only 1 pw guess per session

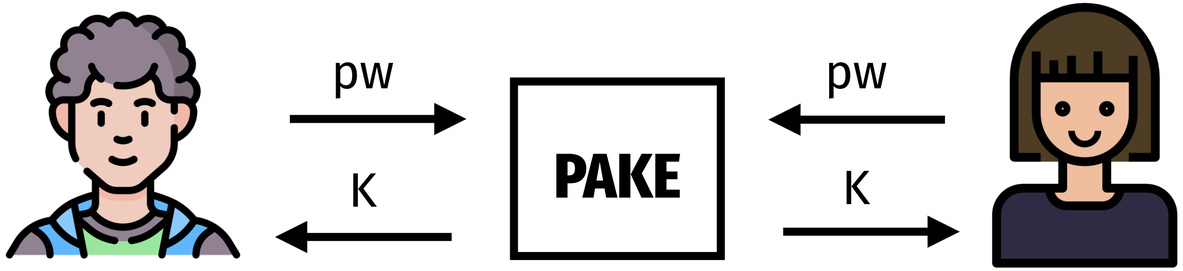


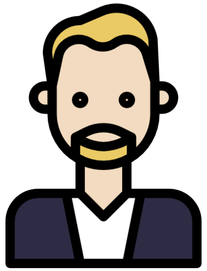
What is a PAKE

Two parties use a password to establish a secure shared secret

A passive adversary cannot derive K

An active adversary has only 1 pw guess per session



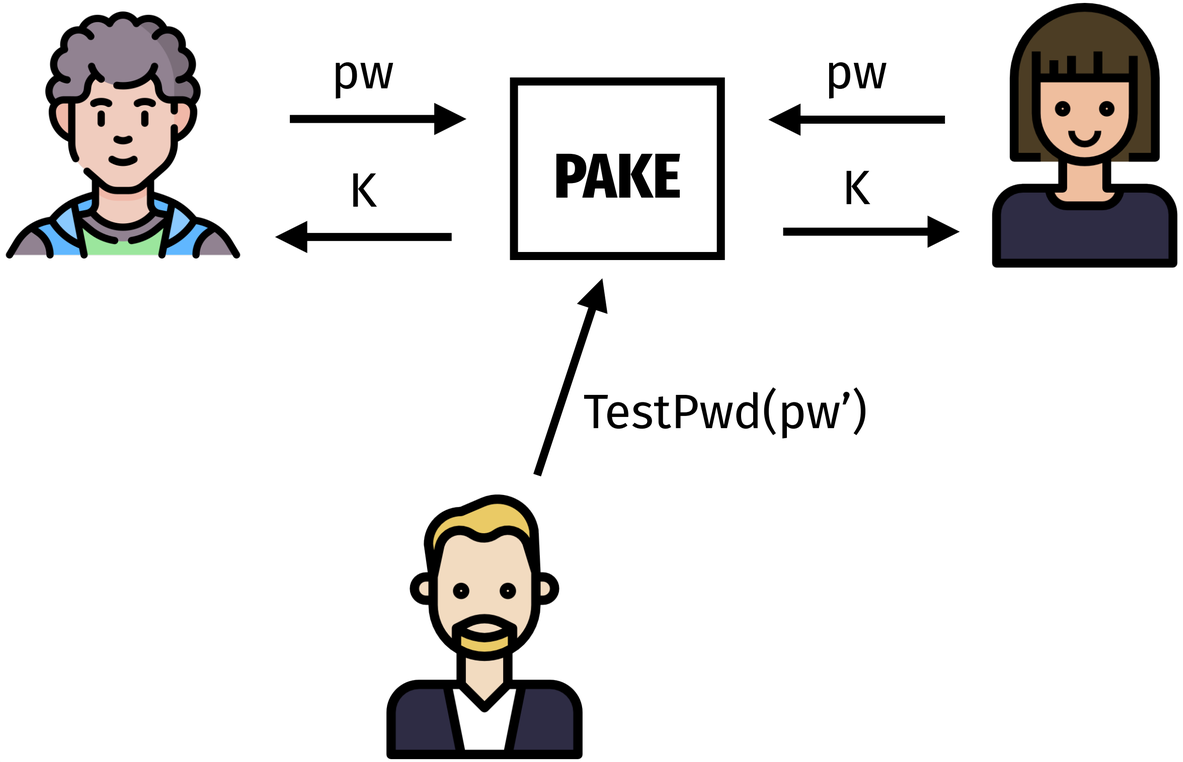


What is a PAKE

Two parties use a password to establish a secure shared secret

A passive adversary cannot derive K

An active adversary has only 1 pw guess per session

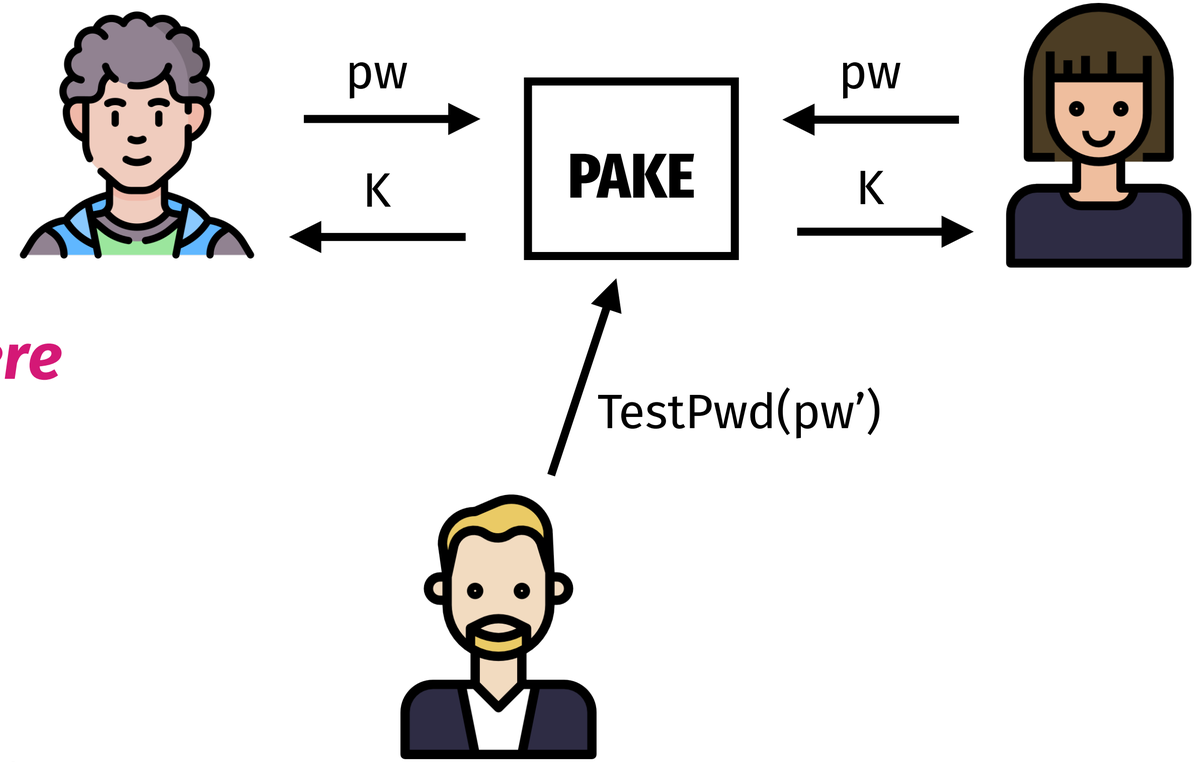


What is a PAKE

Two parties use a password to establish a secure shared secret

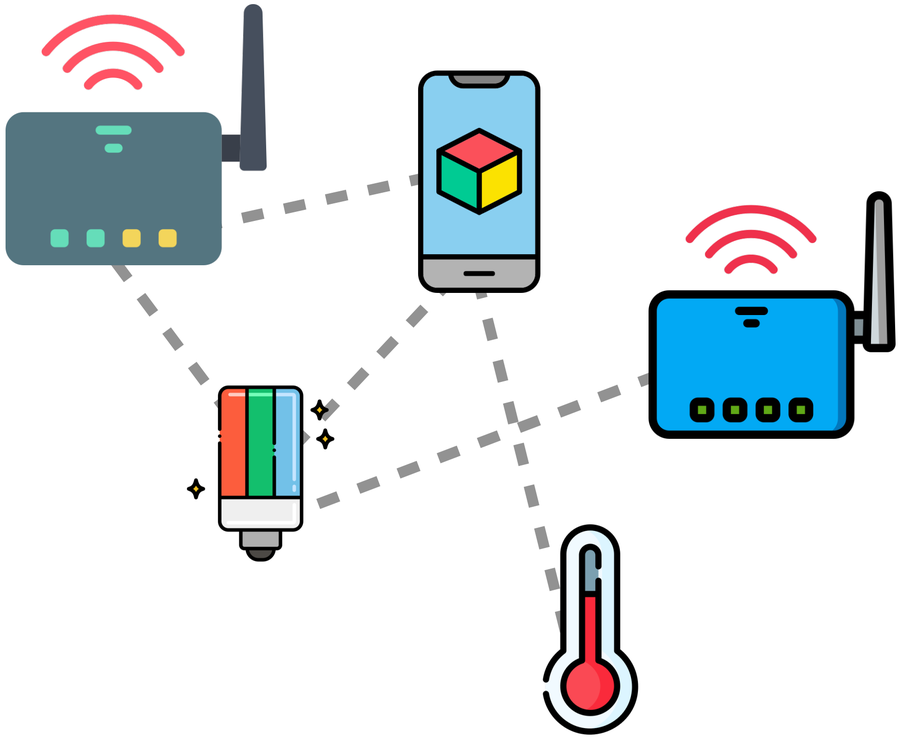
A passive adversary cannot derive K

An active adversary has only 1 pw guess per session



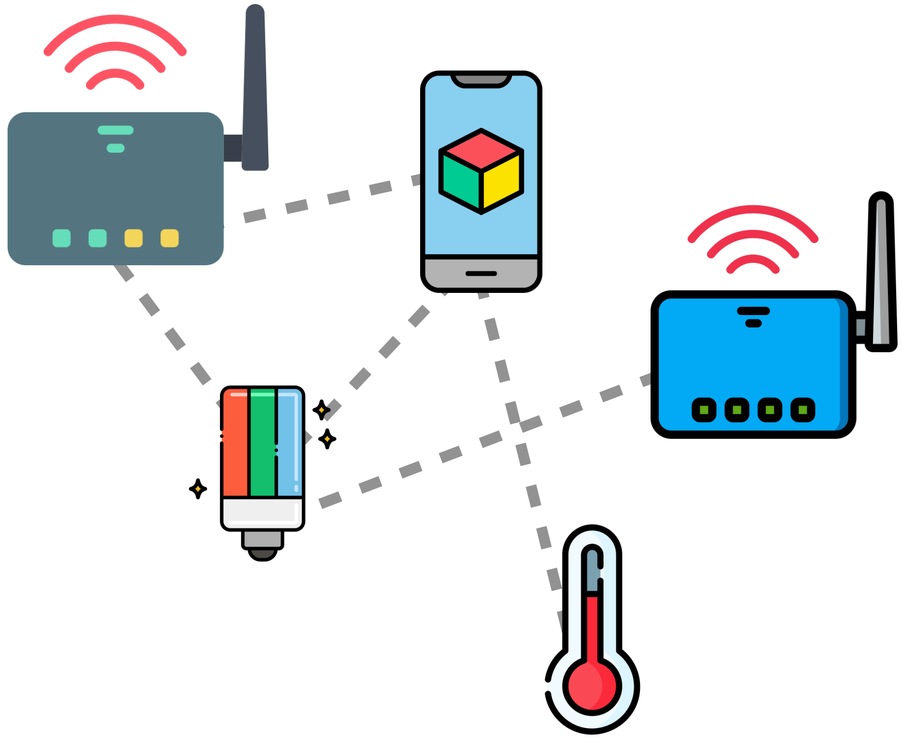
Weird! Cryptographic statements where nothing is high entropy!

PAKEs in IoT



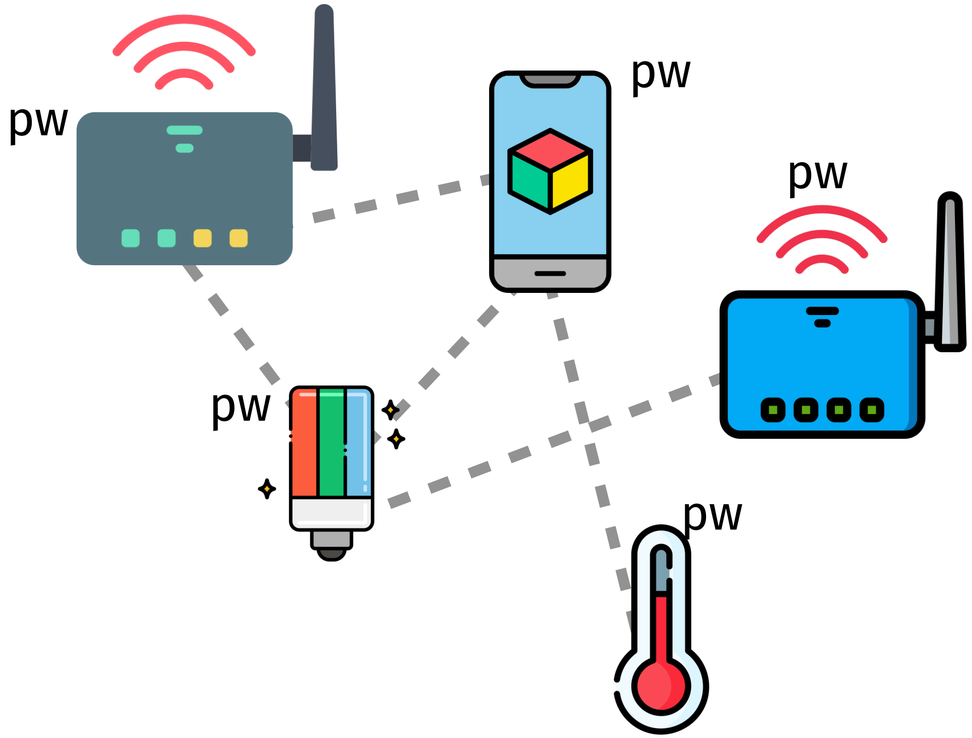
PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password



PAKEs in IoT

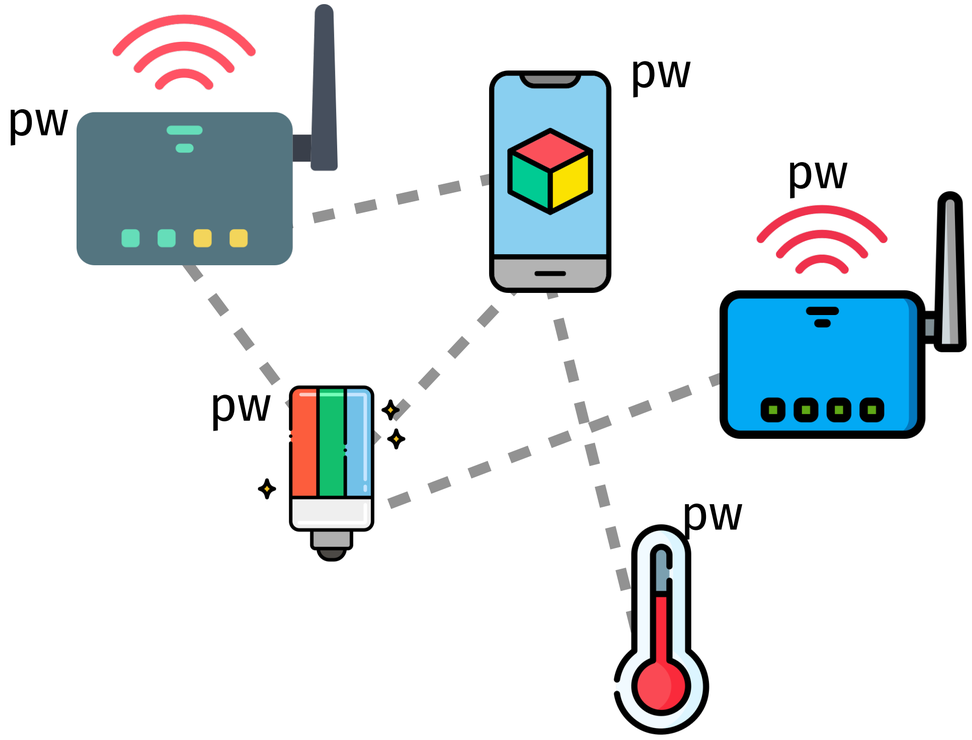
Some IoT protocols do authentication/ key exchange using using just a password



PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

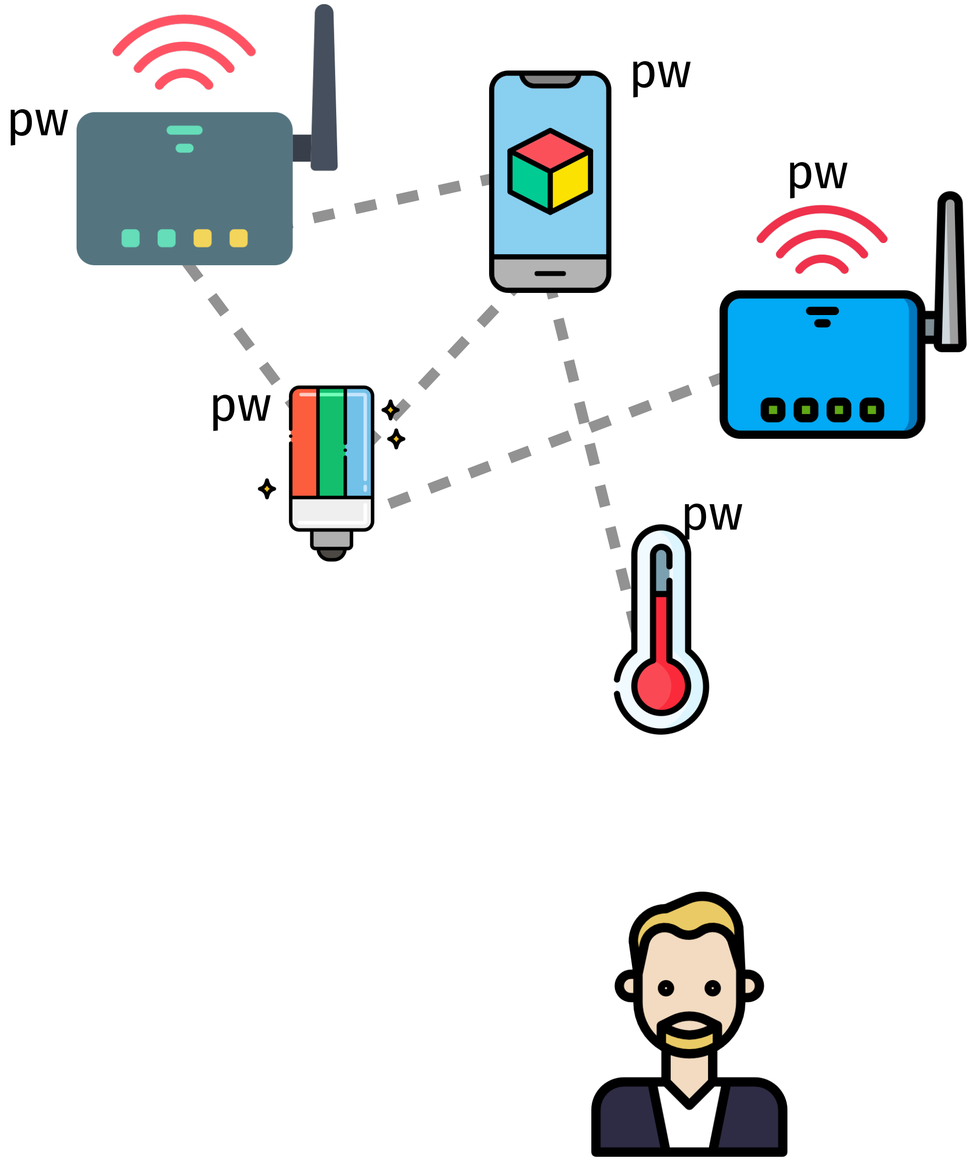
Problem: catastrophic impersonation



PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

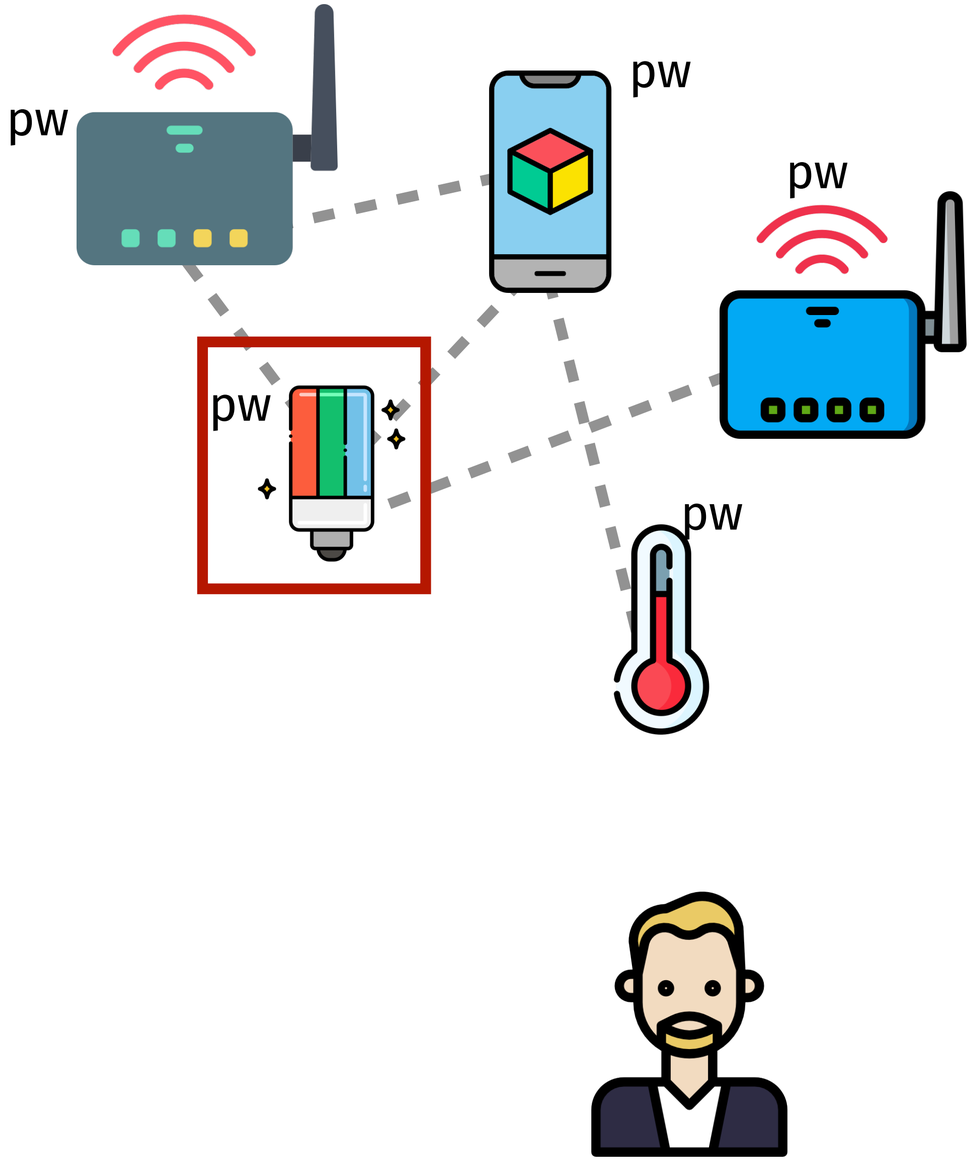
Problem: catastrophic impersonation



PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

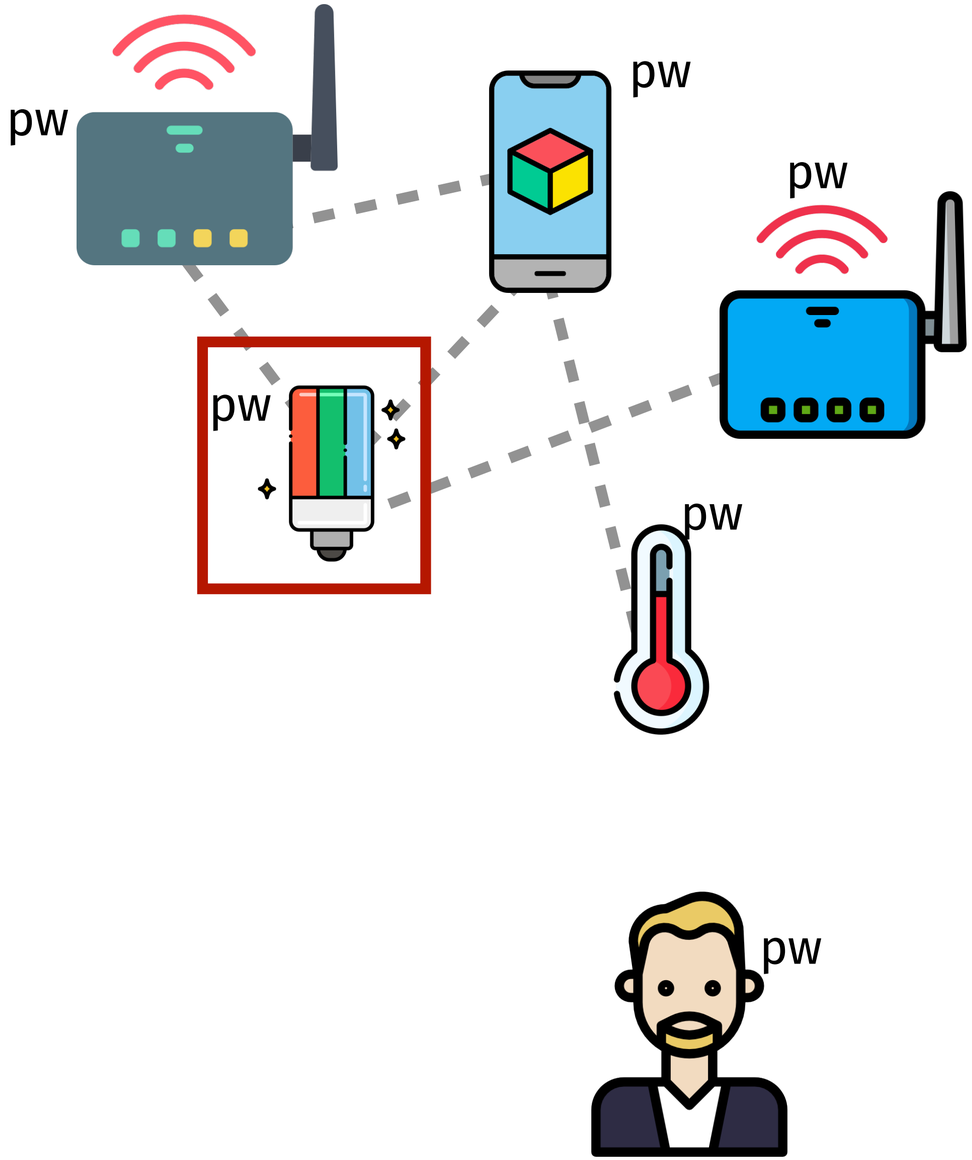
Problem: catastrophic impersonation



PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

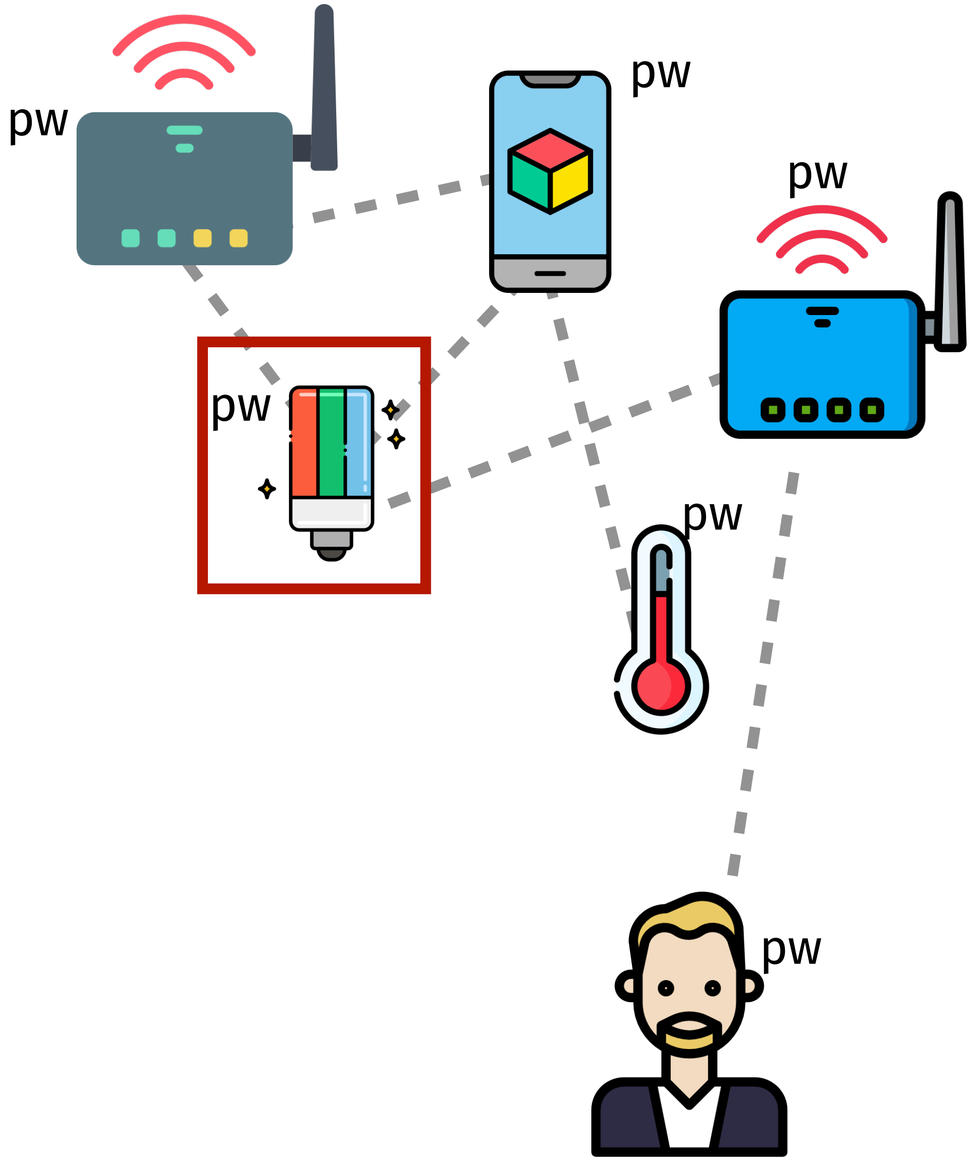
Problem: catastrophic impersonation



PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

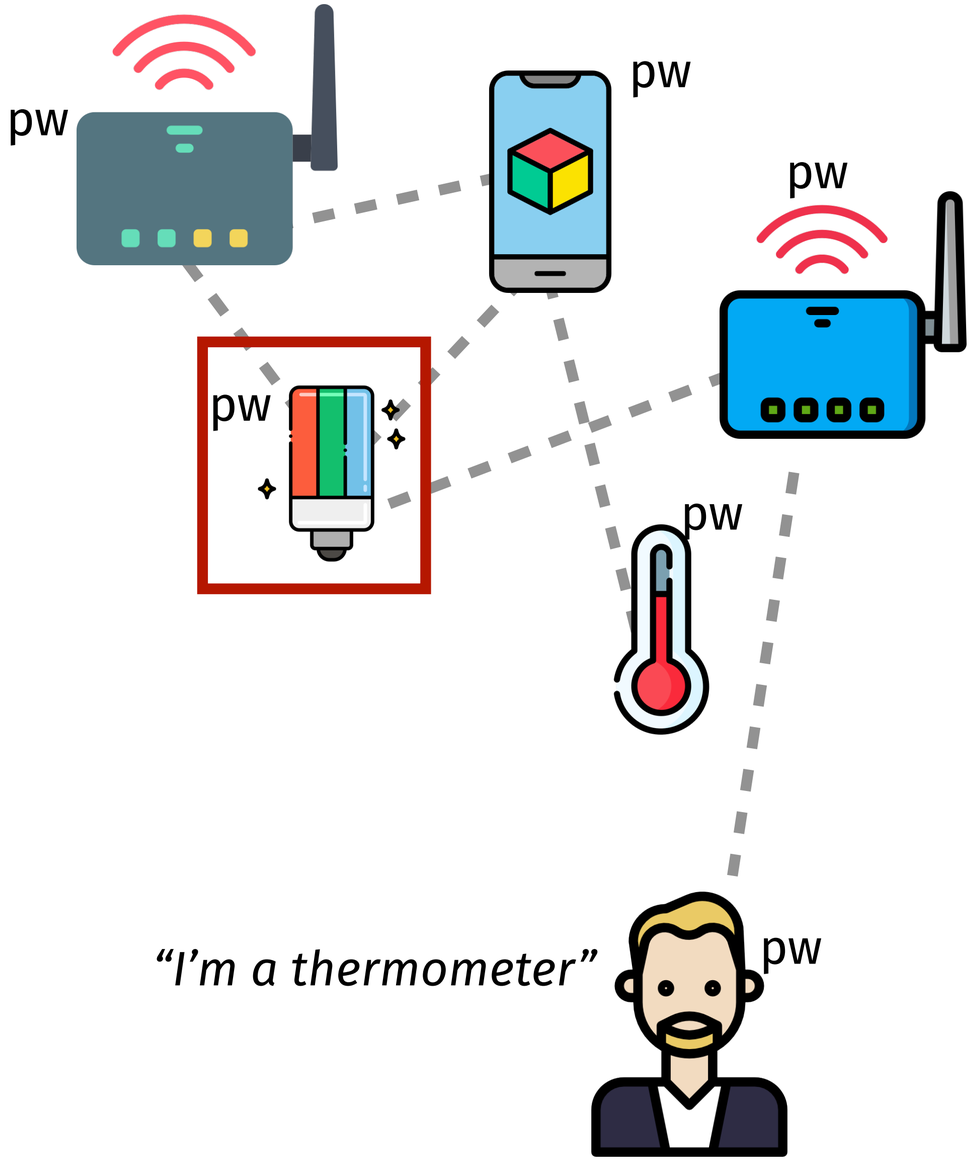
Problem: catastrophic impersonation



PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

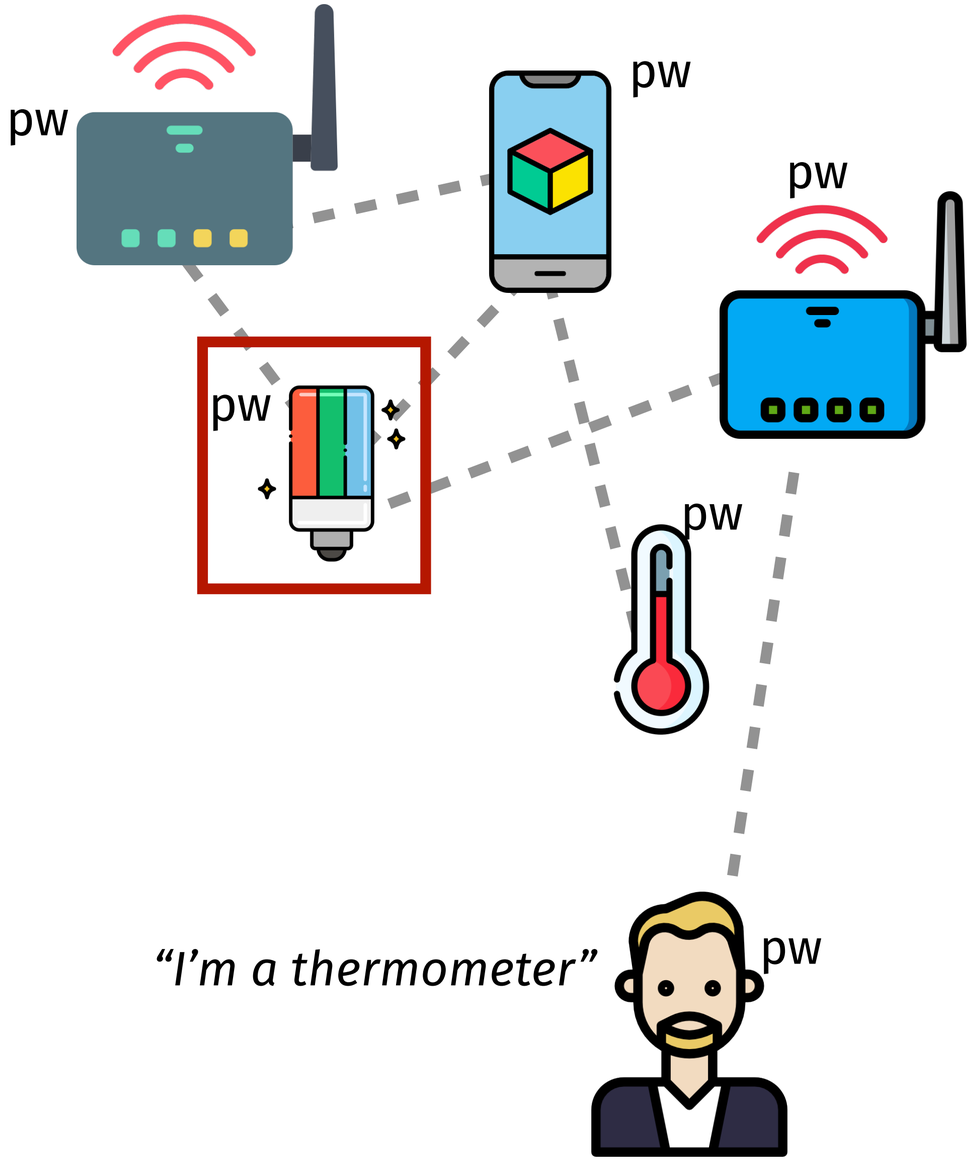


PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution:

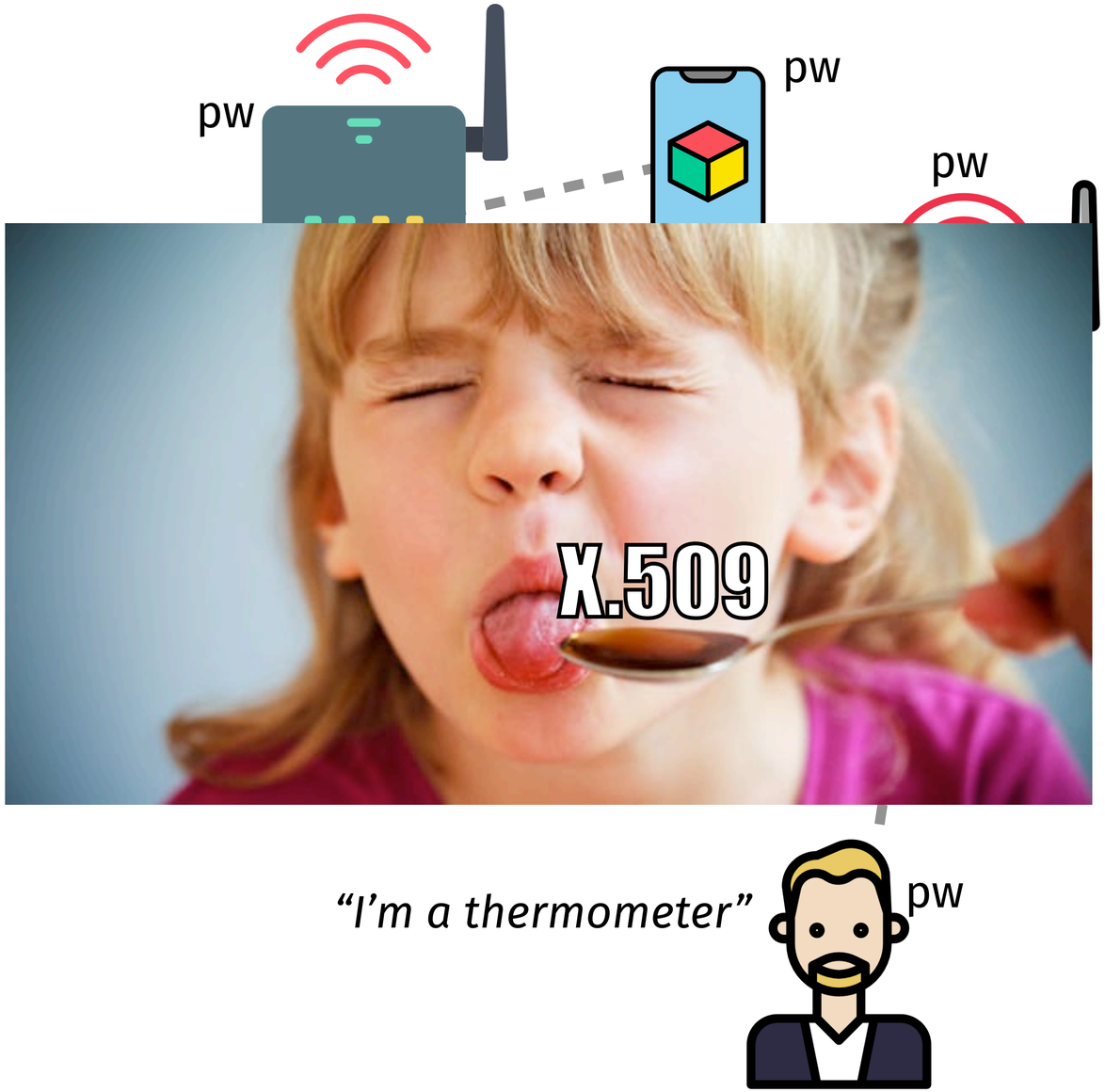


PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution:

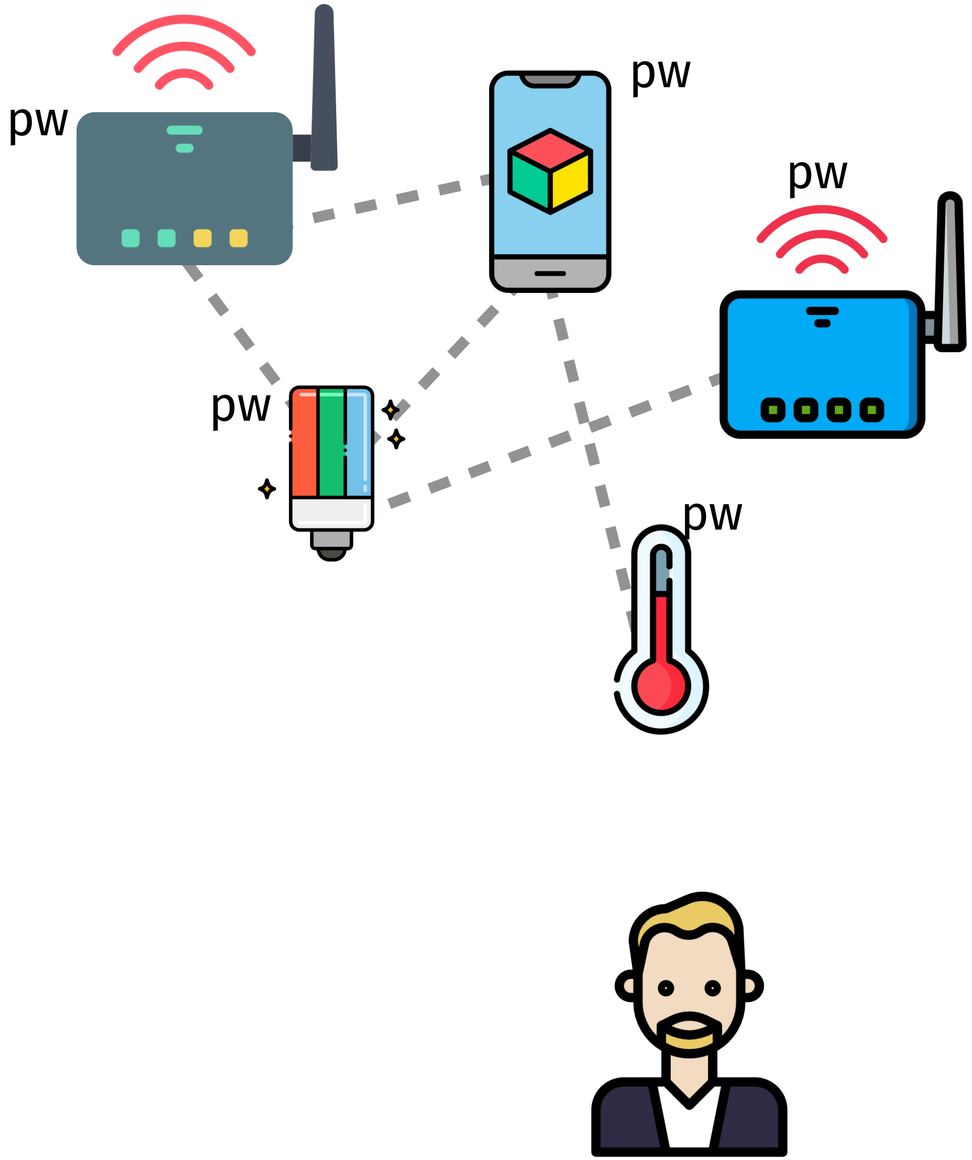


PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**

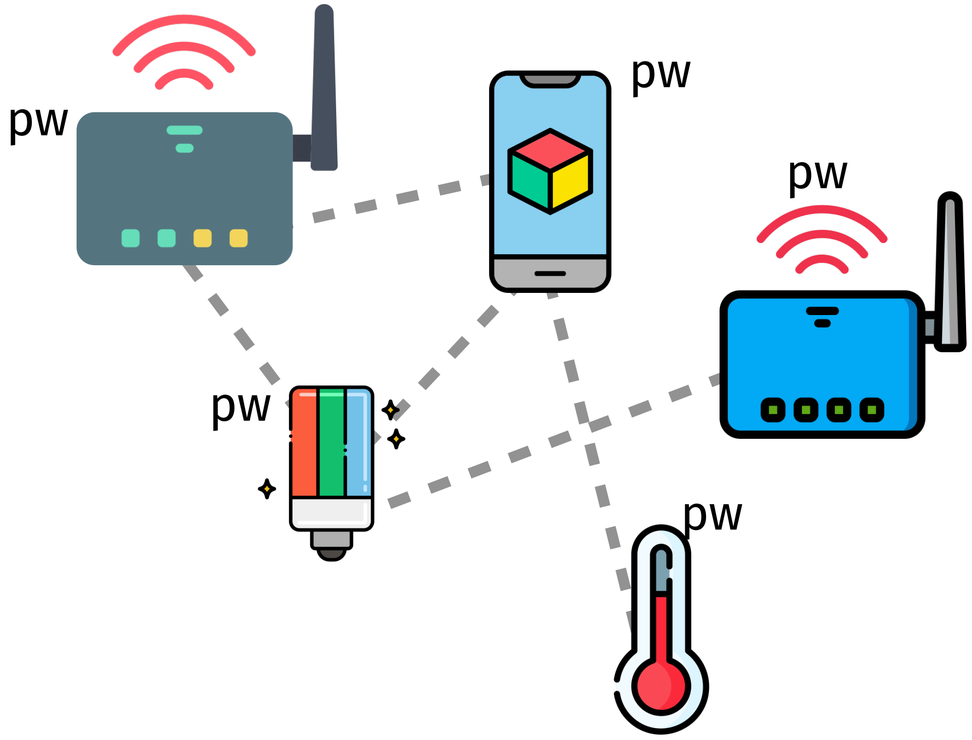


PAKEs in IoT

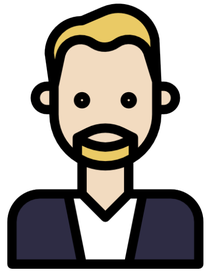
Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**



$pwfile_{id} = f(pw, ID)$

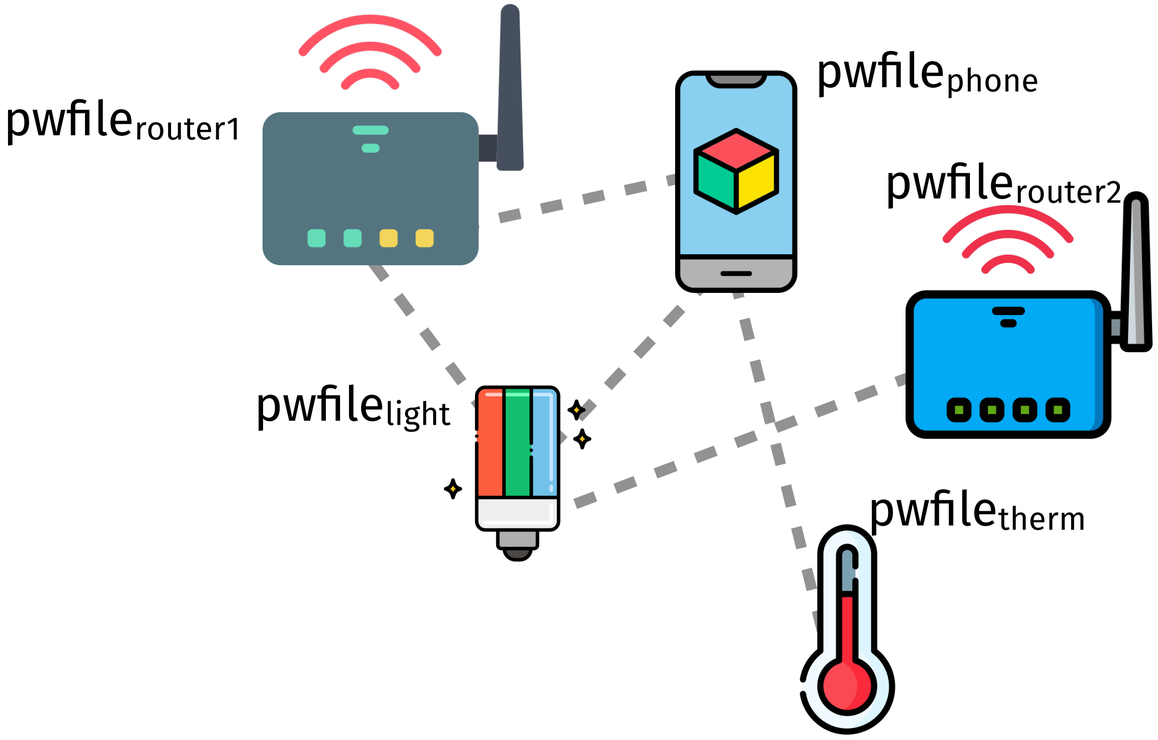


PAKEs in IoT

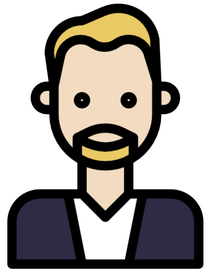
Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**



$\mathbf{pwfile_{id} = f(pw, ID)}$

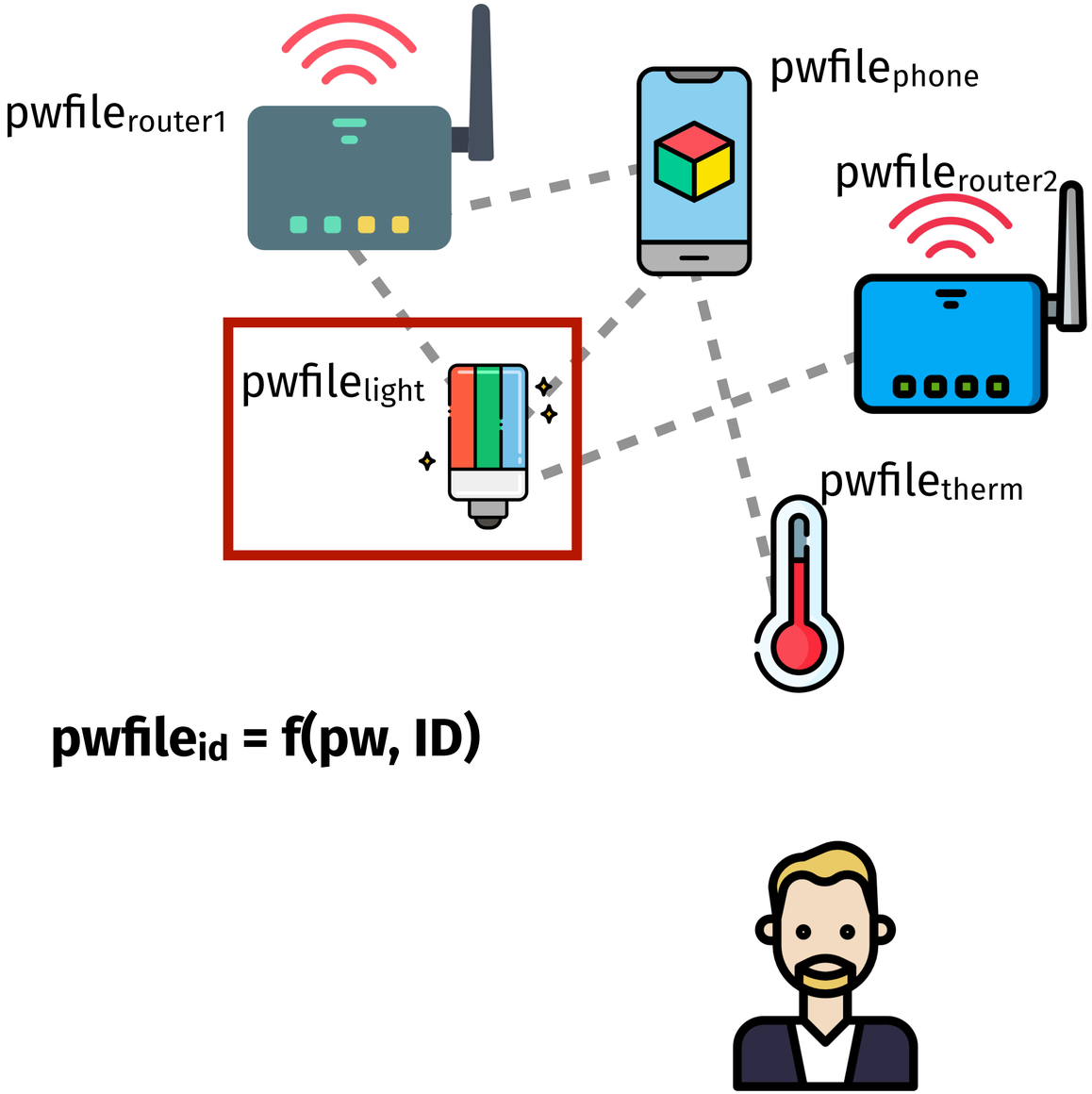


PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**



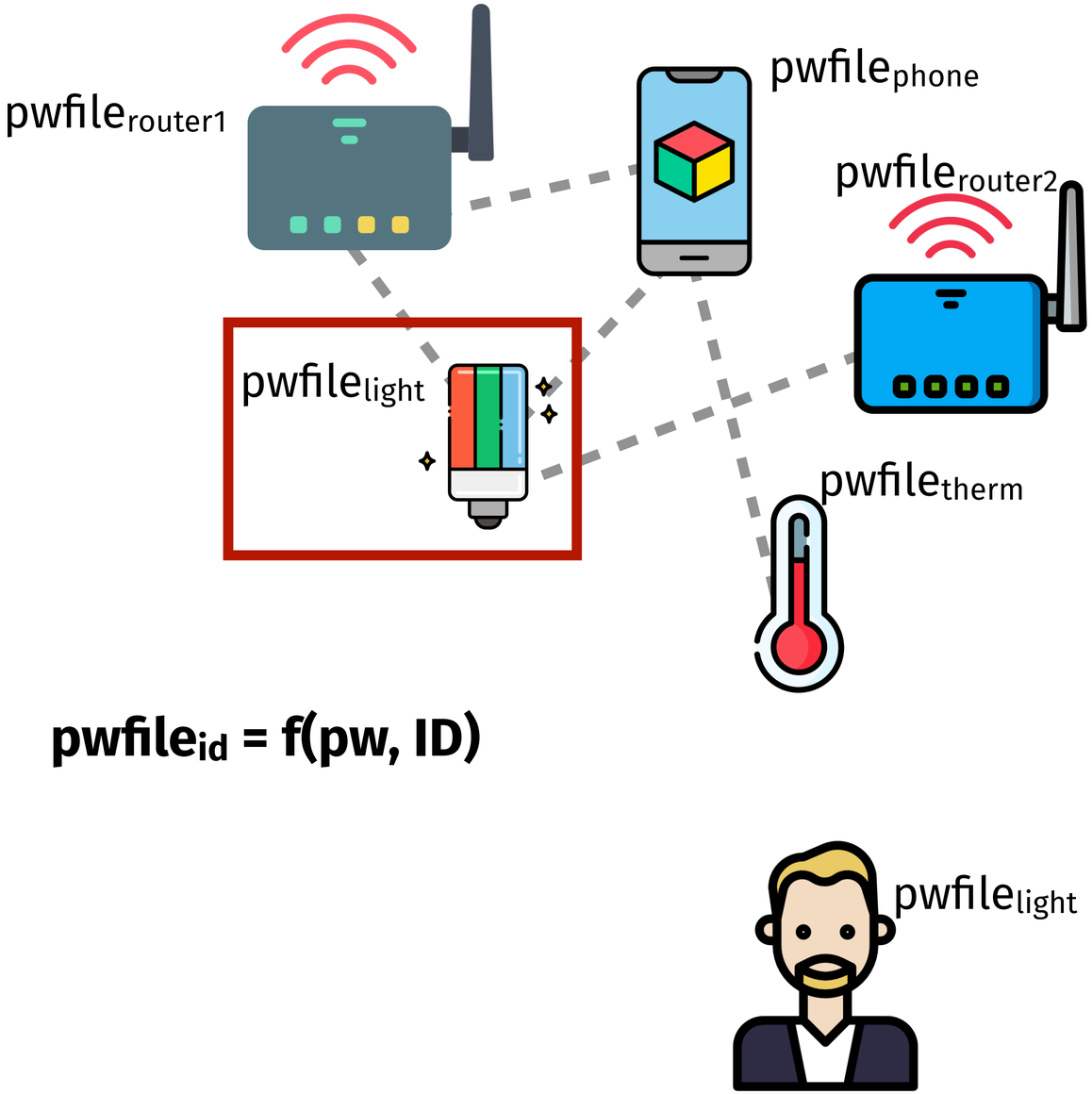
pwfile$_{id}$ = f(pw, ID)

PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**



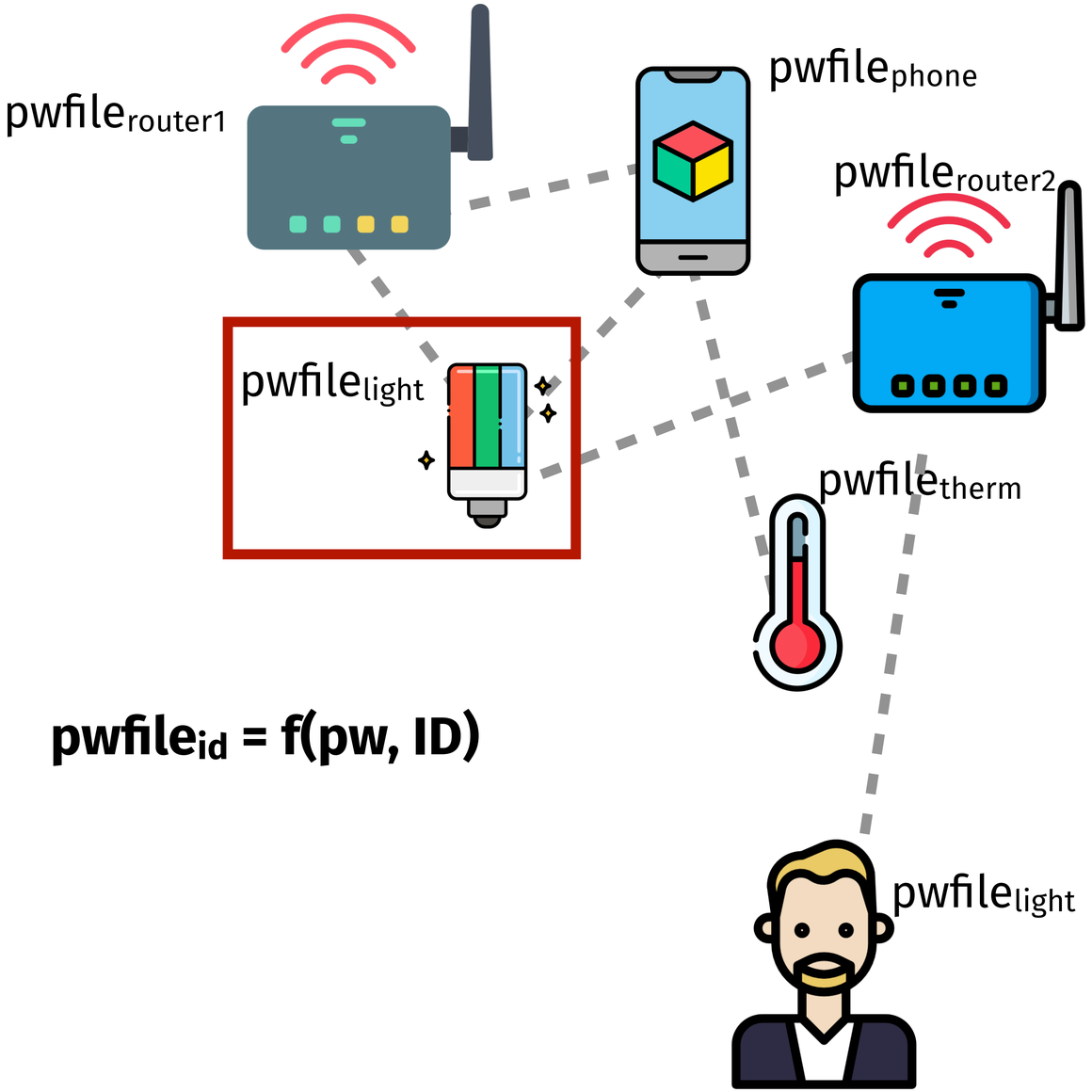
pwfile$_{id}$ = f(pw, ID)

PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**



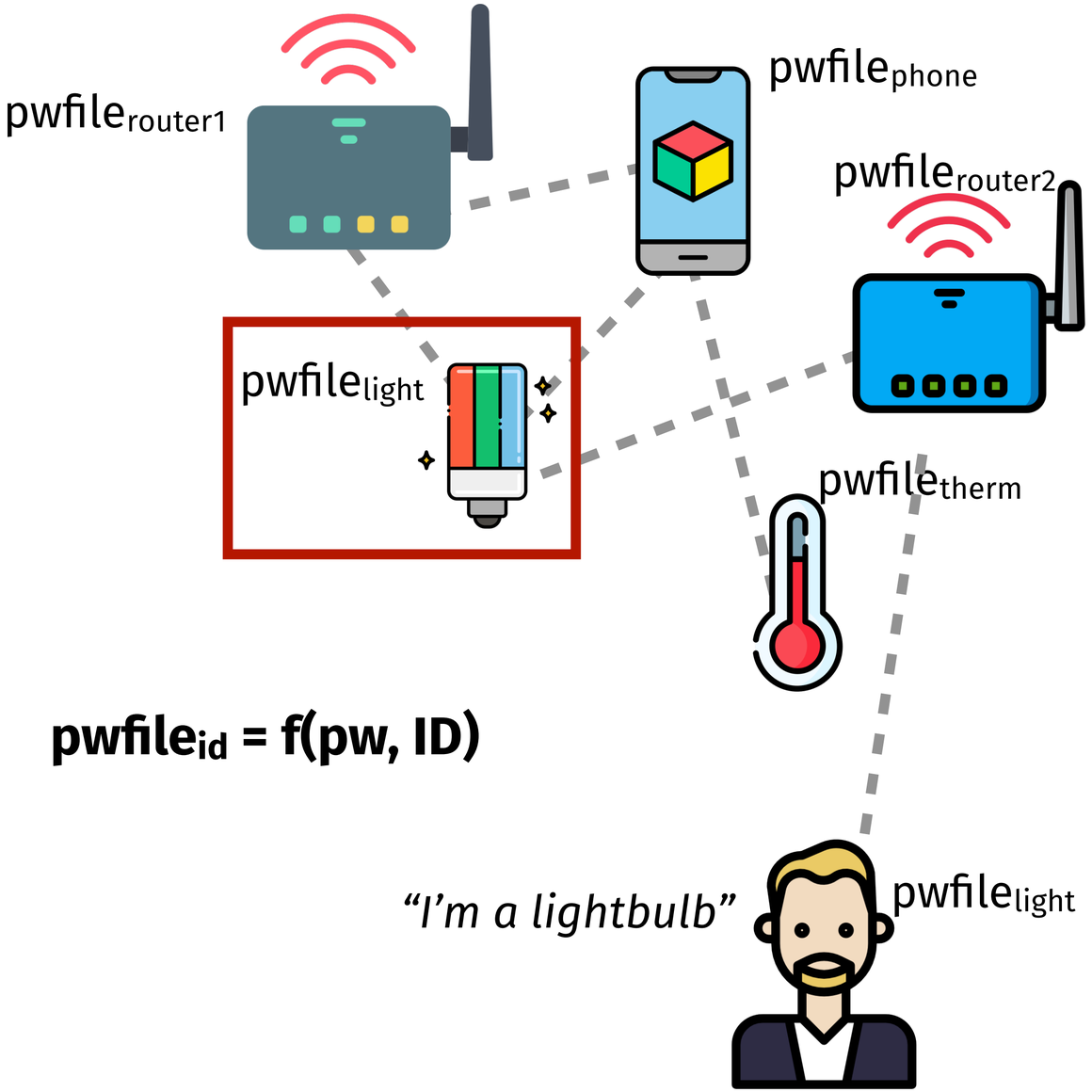
pwfile$_{id}$ = f(pw, ID)

PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**



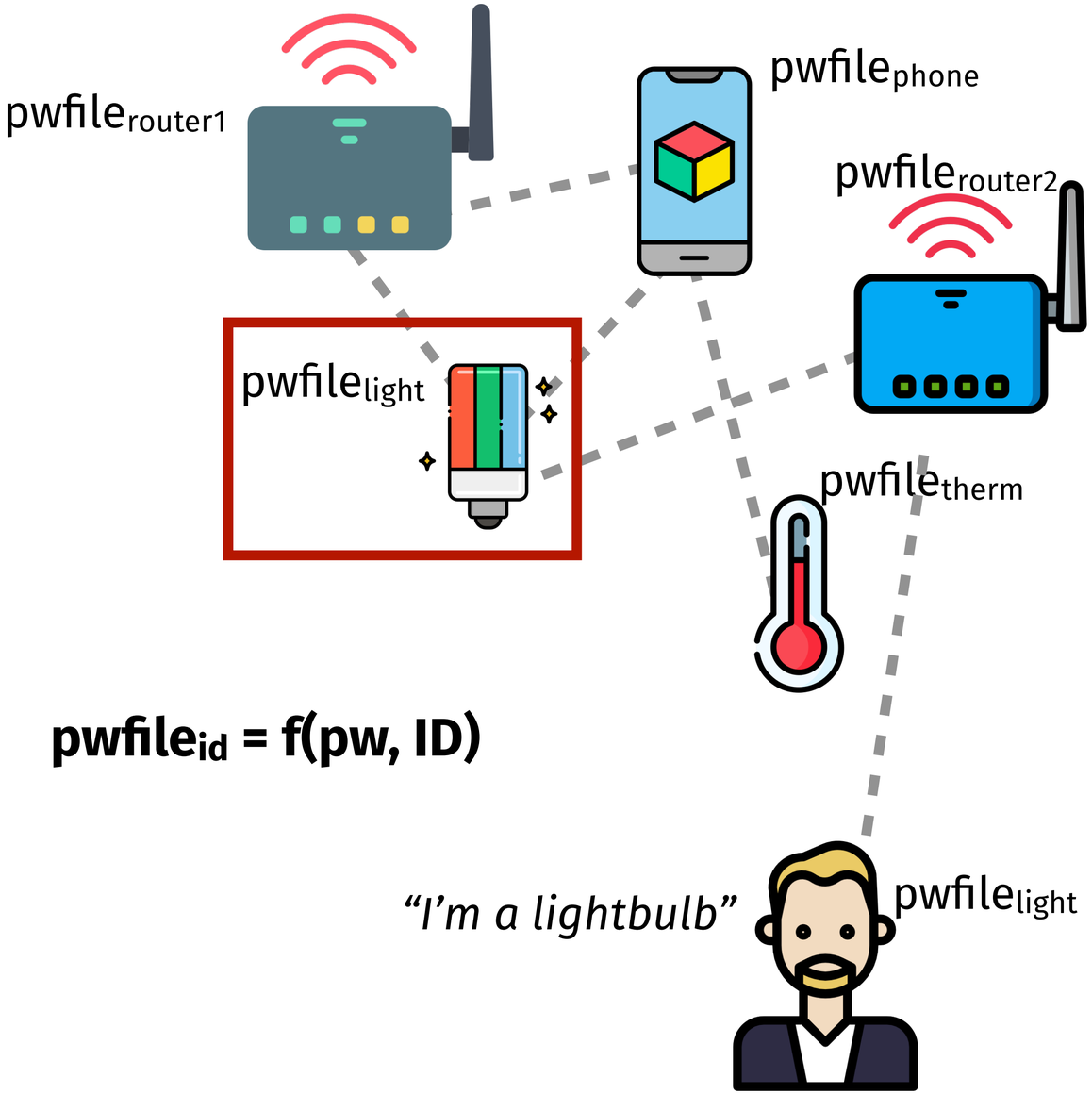
PAKEs in IoT

Some IoT protocols do authentication/ key exchange using using just a password

Problem: catastrophic impersonation

Solution: **bind** the password to the device identity and **delete** the password. Now **every device has a unique pwfile**

This is called an **identity-binding PAKE** (iPAKE) [CNPR22]



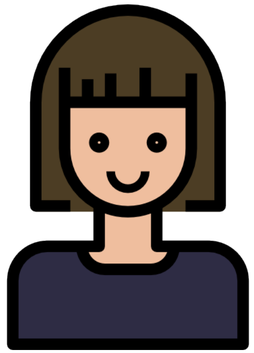
Existing iPAKEs

Existing iPAKEs

[CNPR22] presents **CHIP** and CRISP

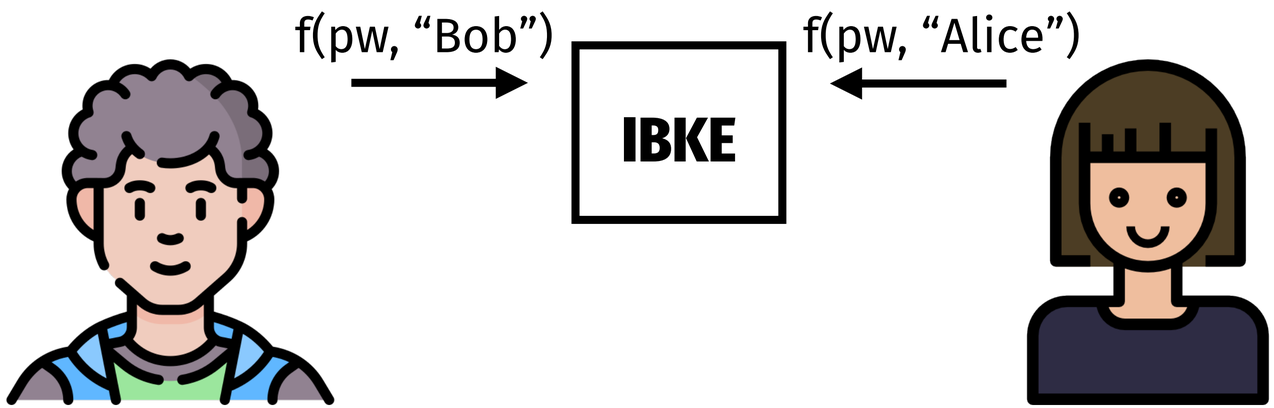
Existing iPAKEs

[CNPR22] presents **CHIP** and CRISP



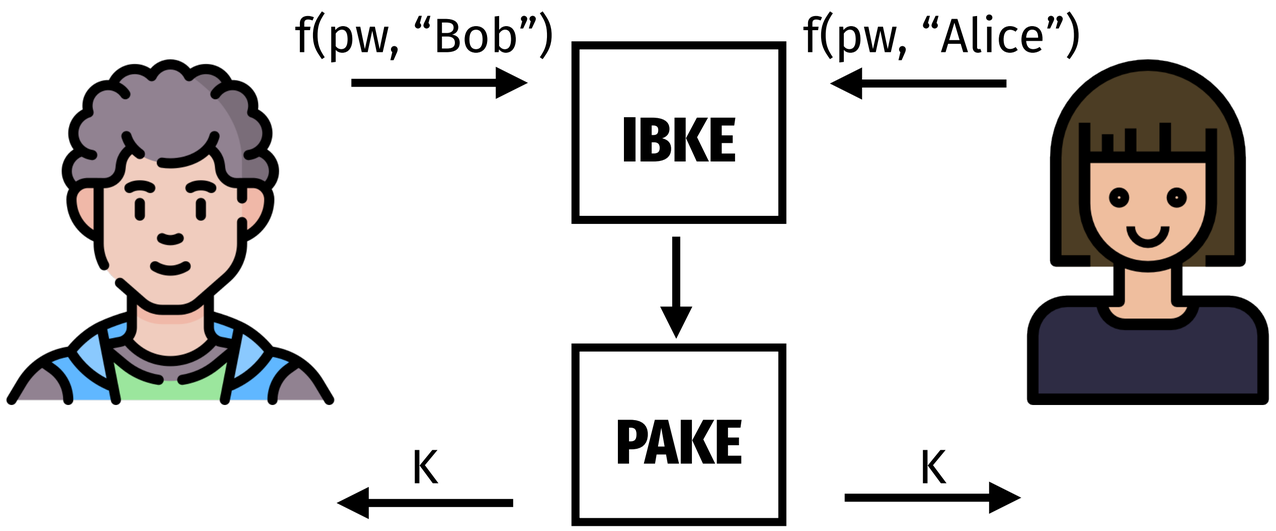
Existing iPAKEs

[CNPR22] presents **CHIP** and CRISP



Existing iPAKEs

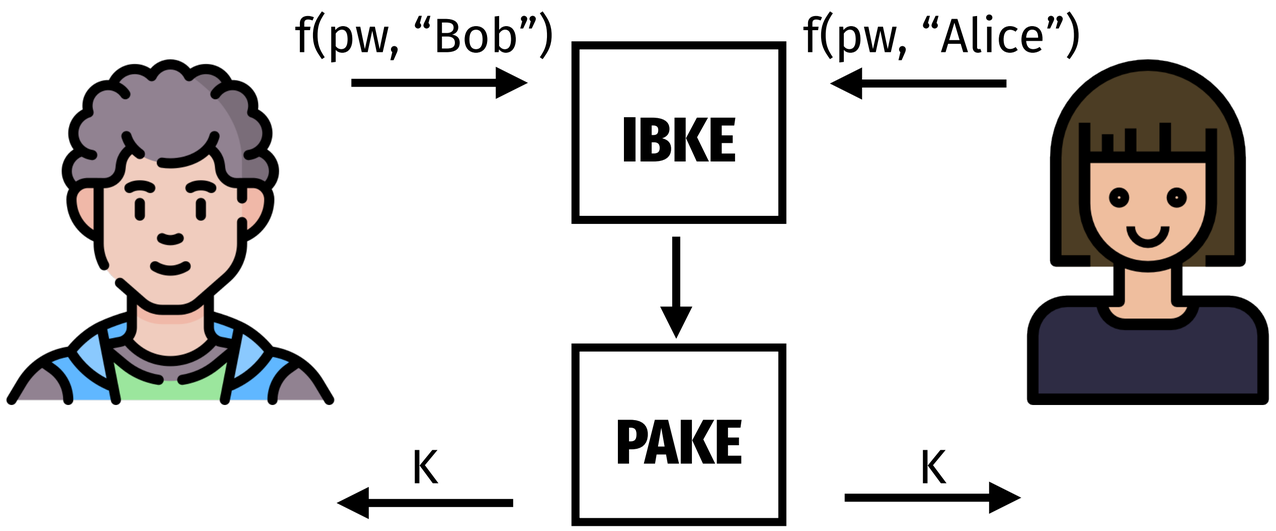
[CNPR22] presents **CHIP** and CRISP



Existing iPAKEs

[CNPR22] presents **CHIP** and CRISP

Both rely on Diffie-Hellman-type assumptions

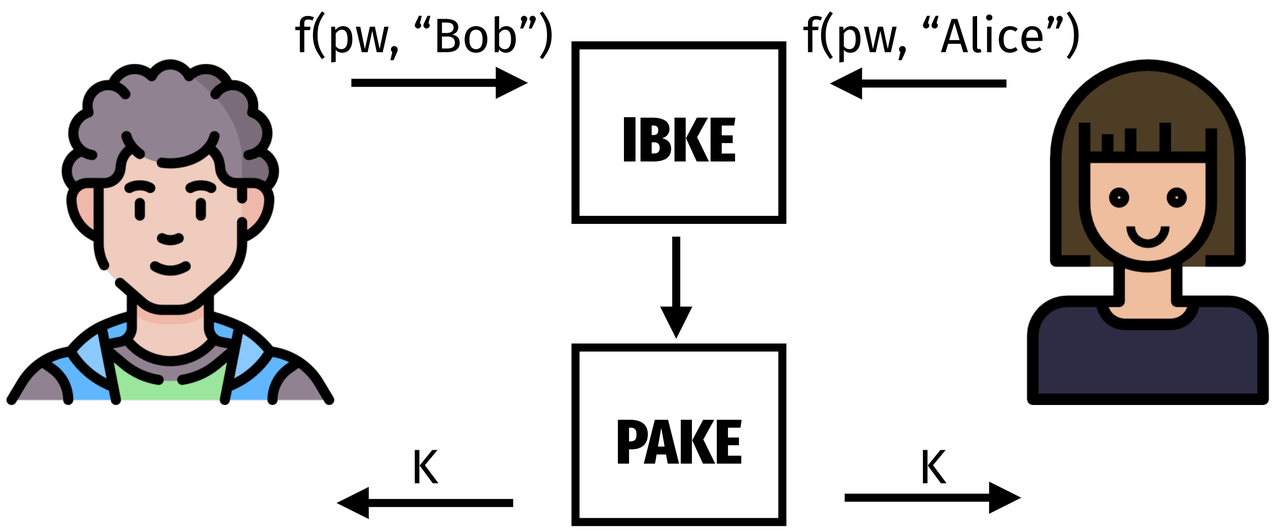


Existing iPAKEs

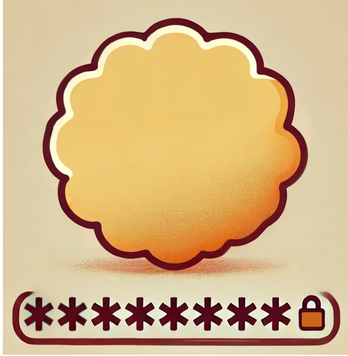
[CNPR22] presents **CHIP** and CRISP

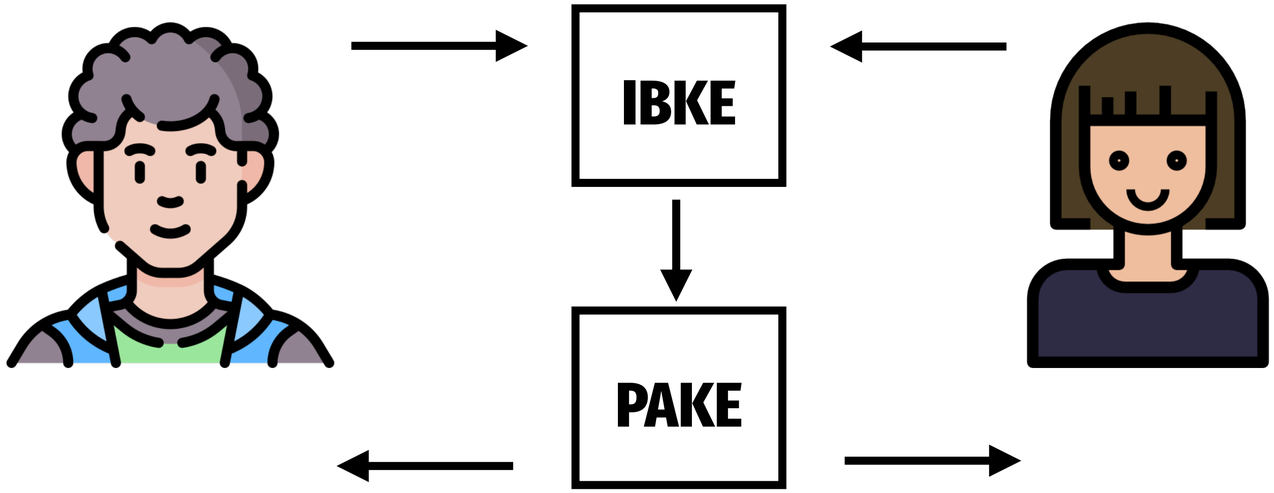
Both rely on Diffie-Hellman-type assumptions

Cannot be made PQ!



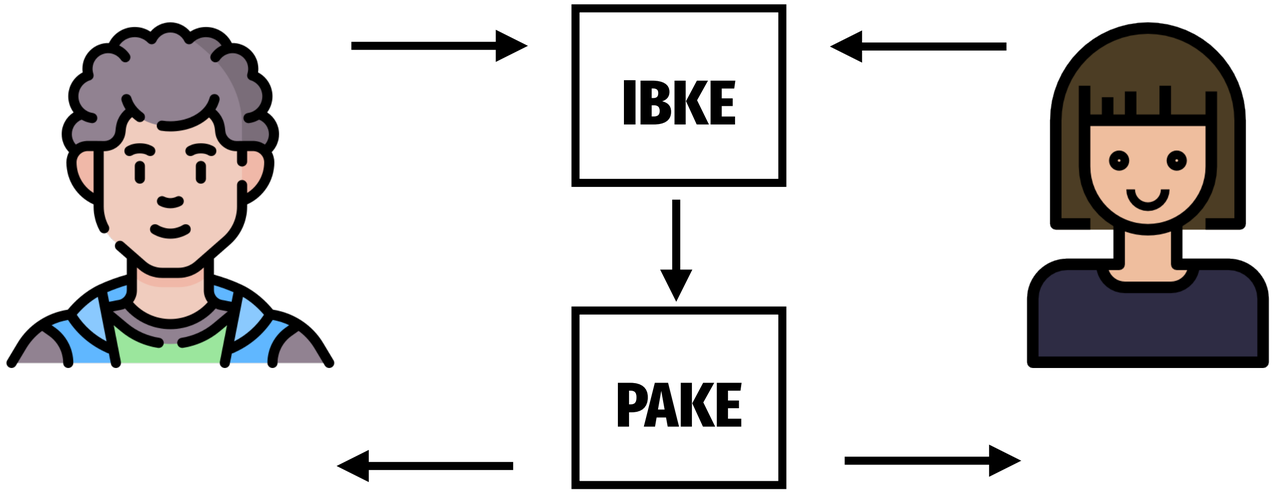
Our contribution





Our contribution

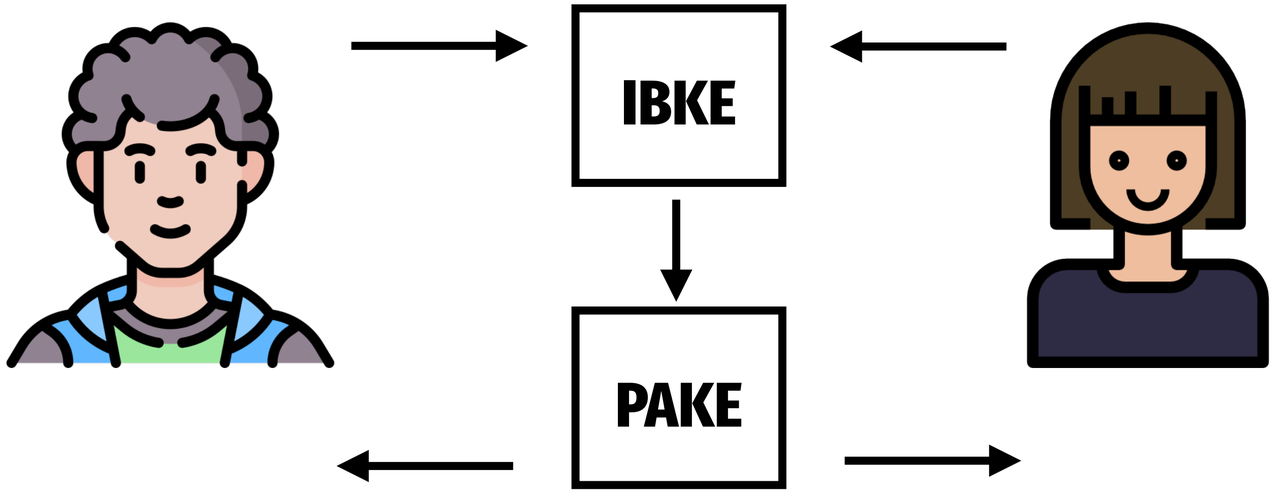
LATKE is the first PQ iPAKE



Our contribution

LATKE is the first PQ iPAKE

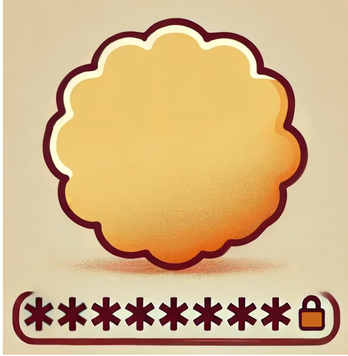
Generic over IBKE and PAKE

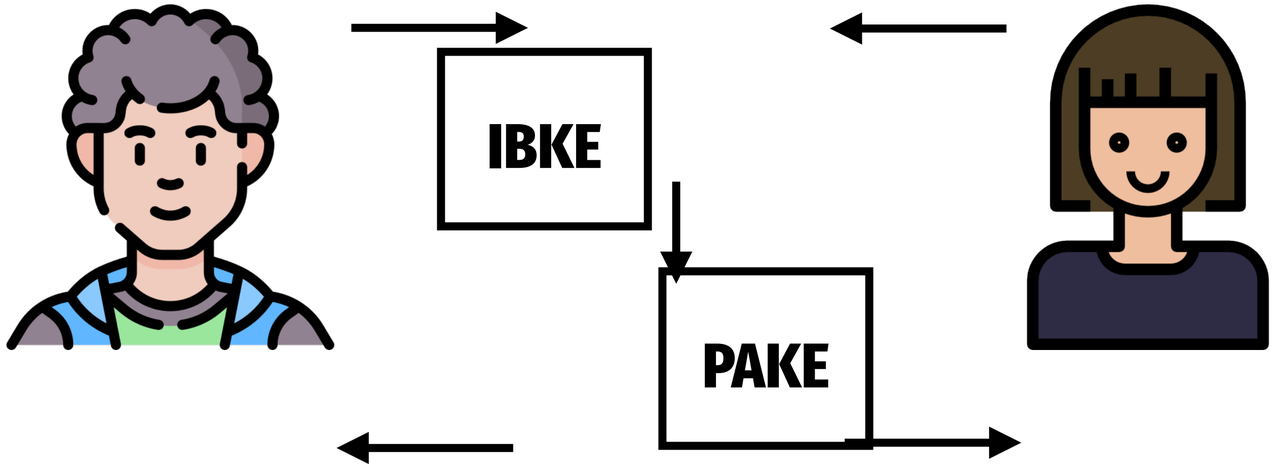


Our contribution

LATKE is the first PQ iPAKE

Generic over IBKE and PAKE

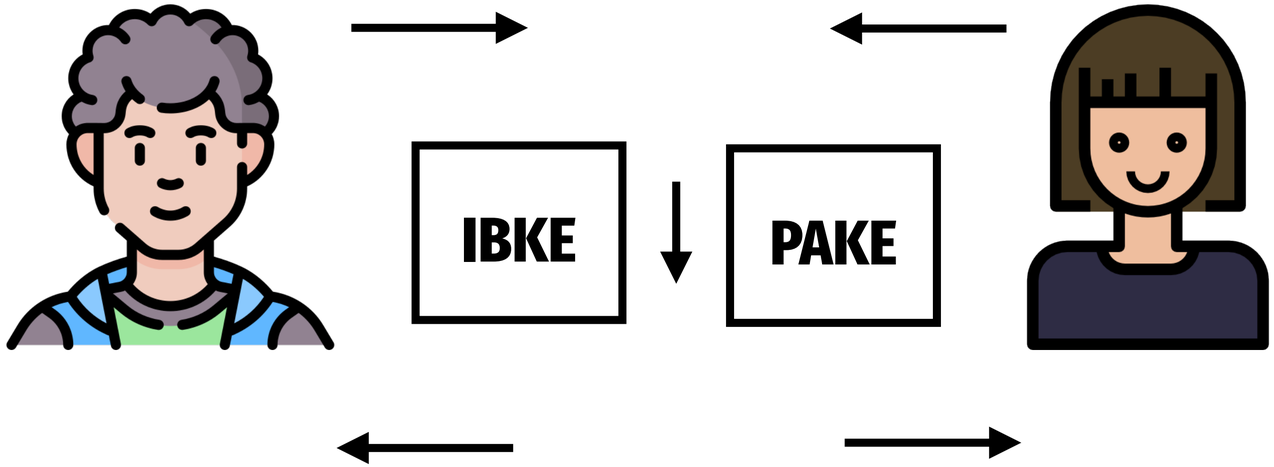




Our contribution

LATKE is the first PQ iPAKE

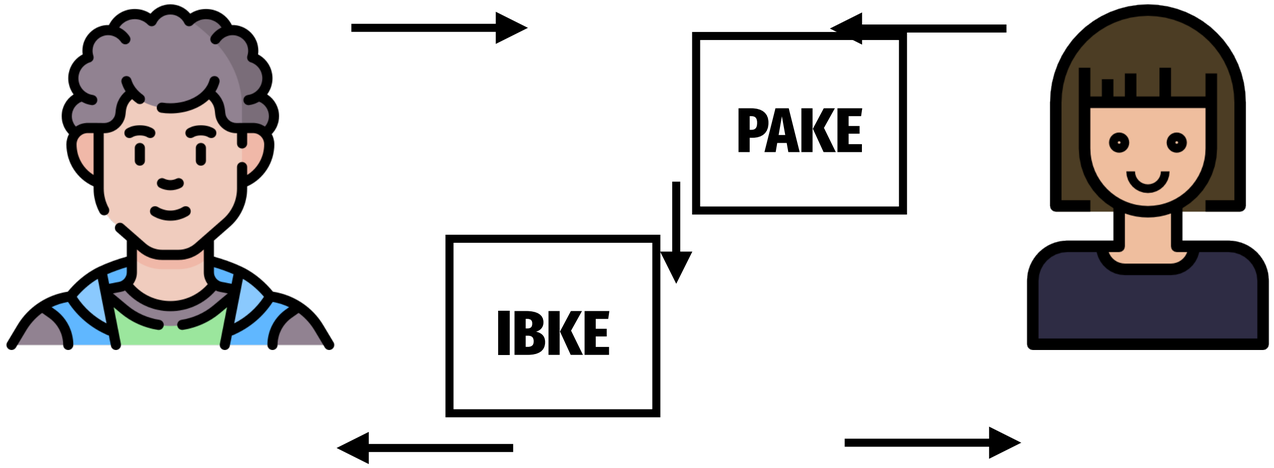
Generic over IBKE and PAKE



Our contribution

LATKE is the first PQ iPAKE

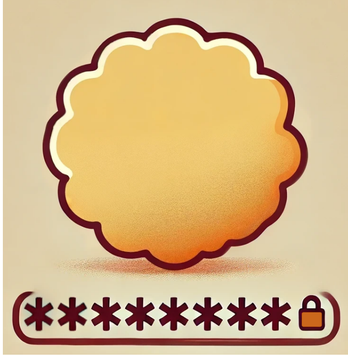
Generic over IBKE and PAKE

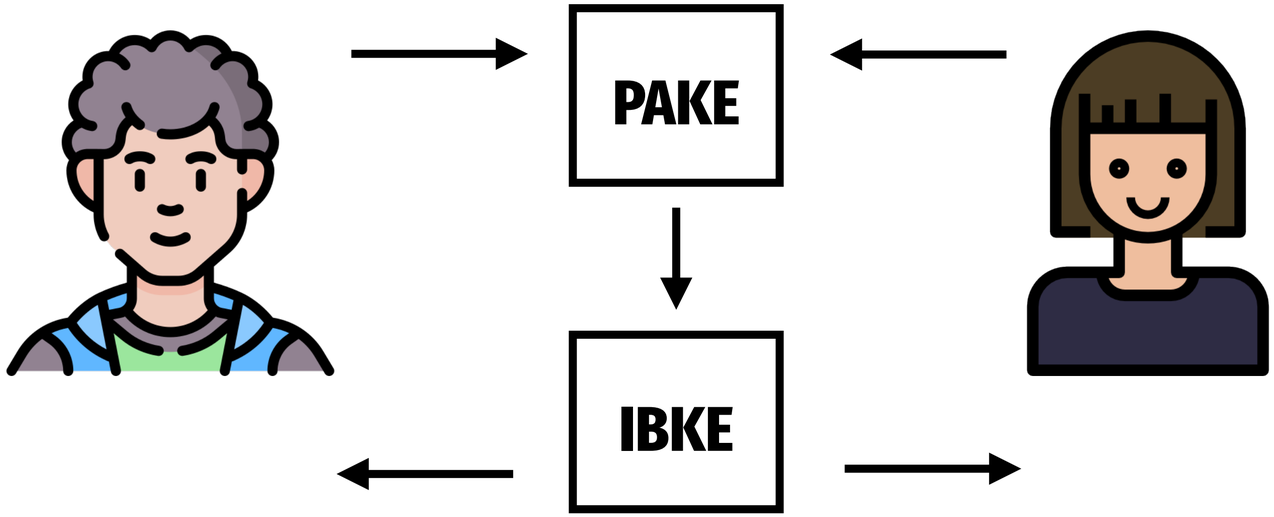


Our contribution

LATKE is the first PQ iPAKE

Generic over IBKE and PAKE



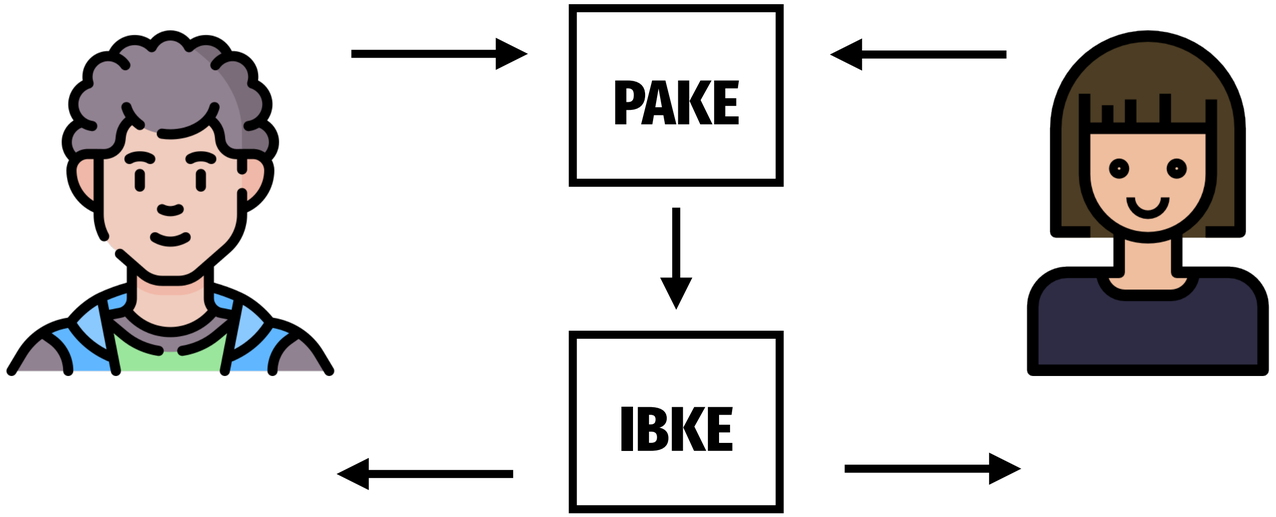


Our contribution

LATKE is the first PQ iPAKE

Generic over IBKE and PAKE

Can be pre- or (plausibly) post-quantum

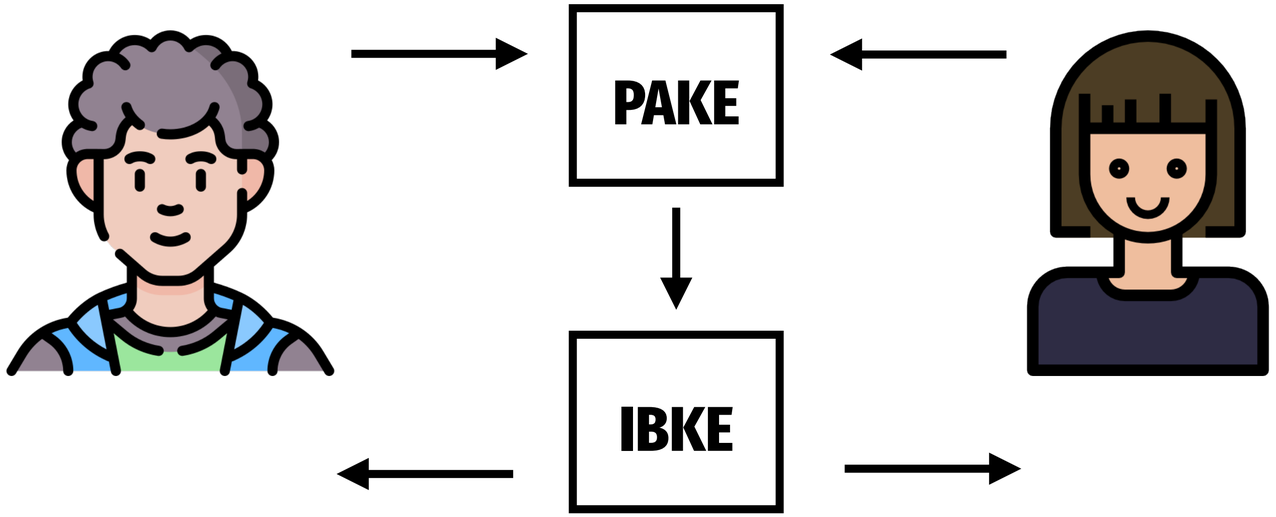


Our contribution

LATKE is the first PQ iPAKE

Generic over IBKE and PAKE

- Can be pre- or (plausibly) post-quantum
- Works in pre-specified and post-specified peer model

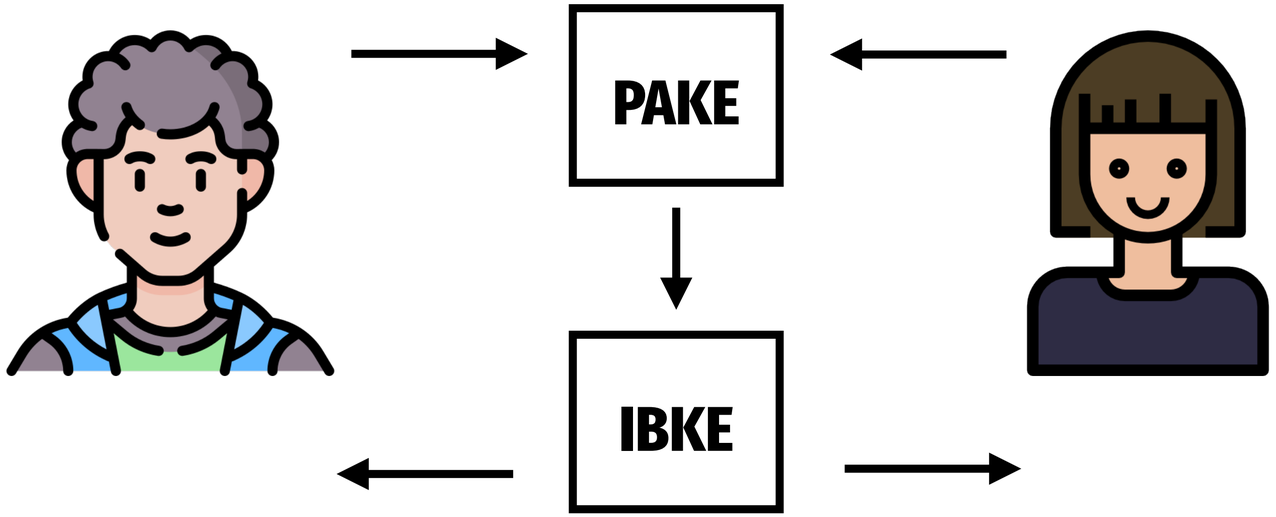


Our contribution

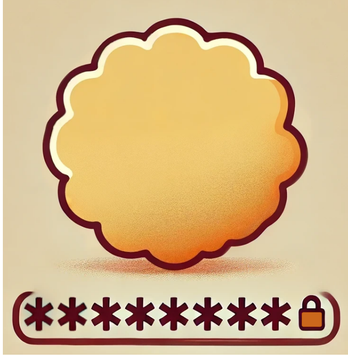
LATKE is the first PQ iPAKE

Generic over IBKE and PAKE

- Can be pre- or (plausibly) post-quantum
- Works in pre-specified and post-specified peer model
- Works with any reasonable key exchange security def (CK, CK_{HMQV}, CK^{+}, eCK)



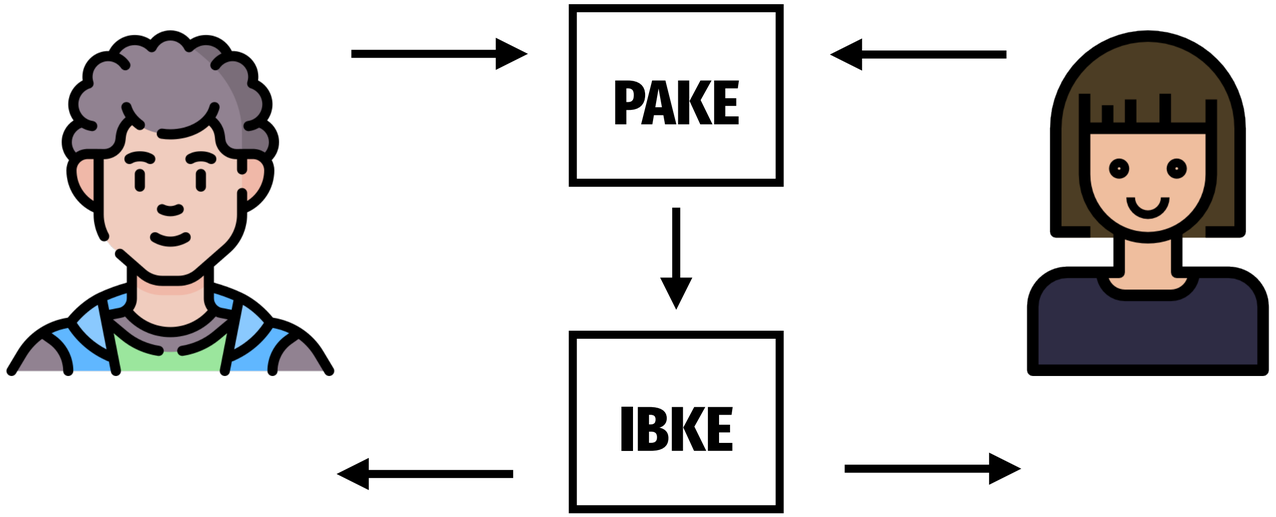
Our contribution



LATKE is the first PQ iPAKE

Generic over IBKE and PAKE

- Can be pre- or (plausibly) post-quantum
- Works in pre-specified and post-specified peer model
- Works with any reasonable key exchange security def (CK, CK_{HMQV}, CK^{+}, eCK)



Post-quantum LATKE is **only 3% slower** than pre-quantum CHIP (ignoring comms costs)

The CHIP iPAKE

The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

The CHIP iPAKE

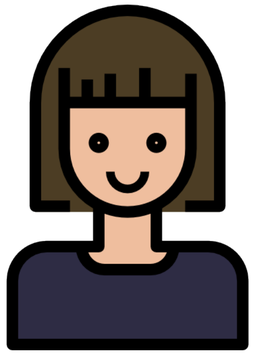
Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

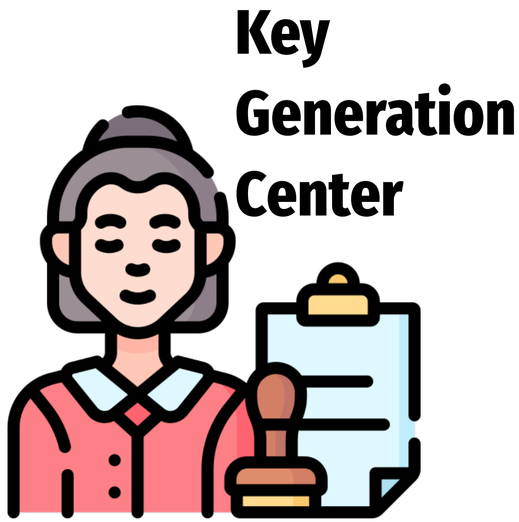
This sounds like identity-based cryptography!

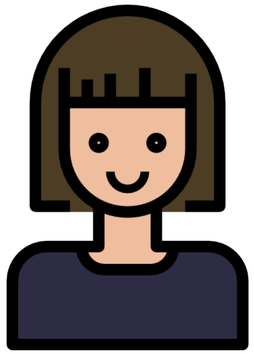


The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

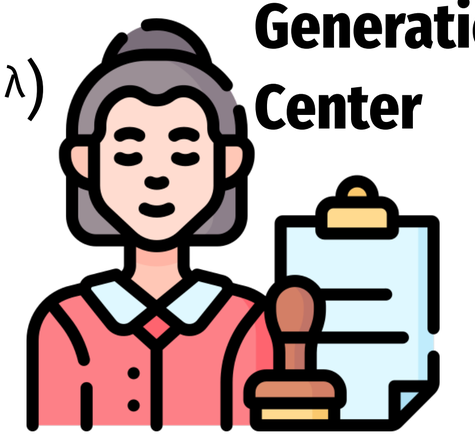
This sounds like identity-based cryptography!





The CHIP iPAKE

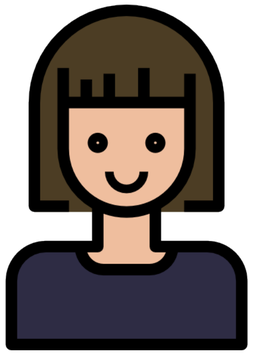
$(msk, mpk) := KeyGen(1^\lambda)$



Key Generation Center

Need to bind an ID to a key (pw) in an irreversible way.

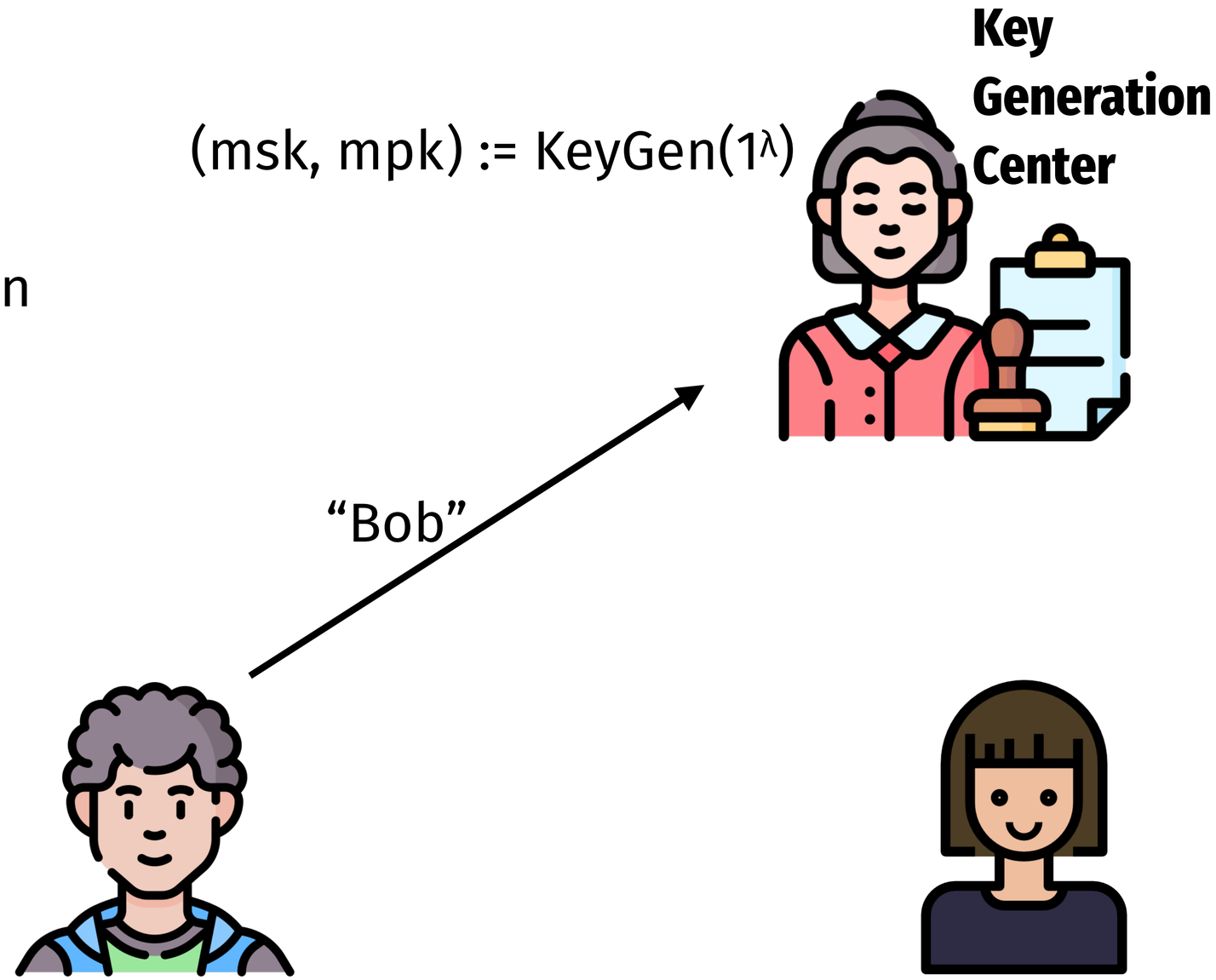
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!



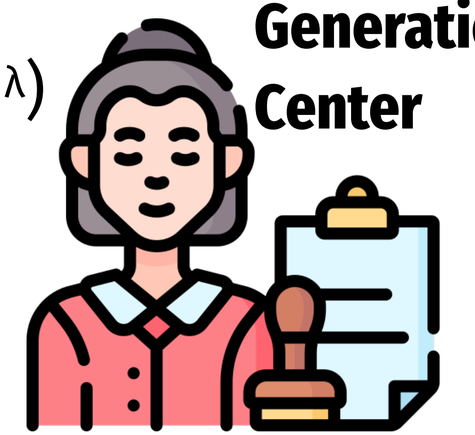
The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

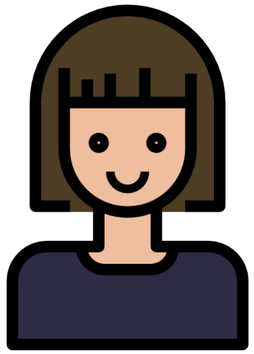
$(msk, mpk) := KeyGen(1^\lambda)$

$sk_B := Extract_{msk}(\text{“Bob”})$



Key Generation Center

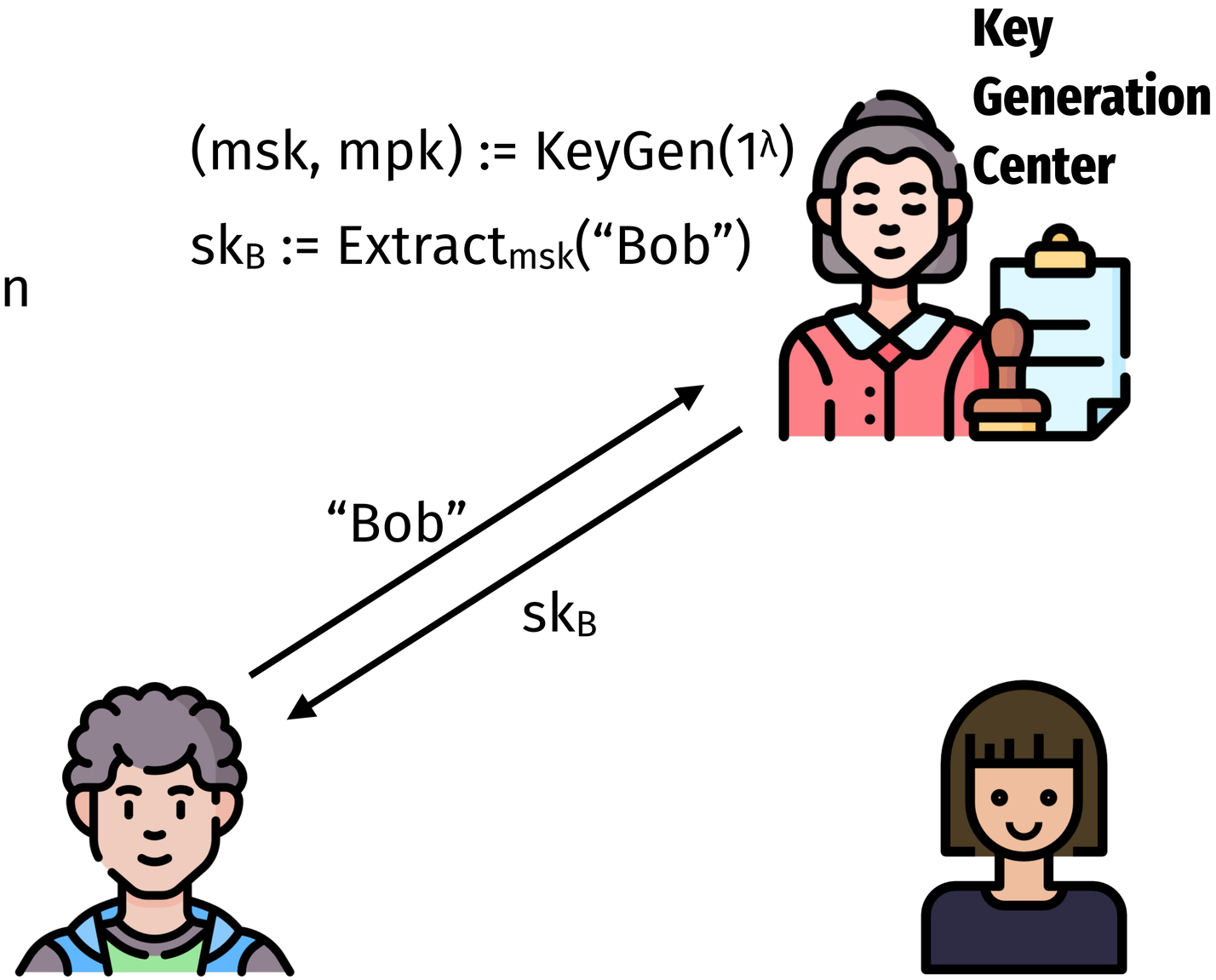
“Bob”



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

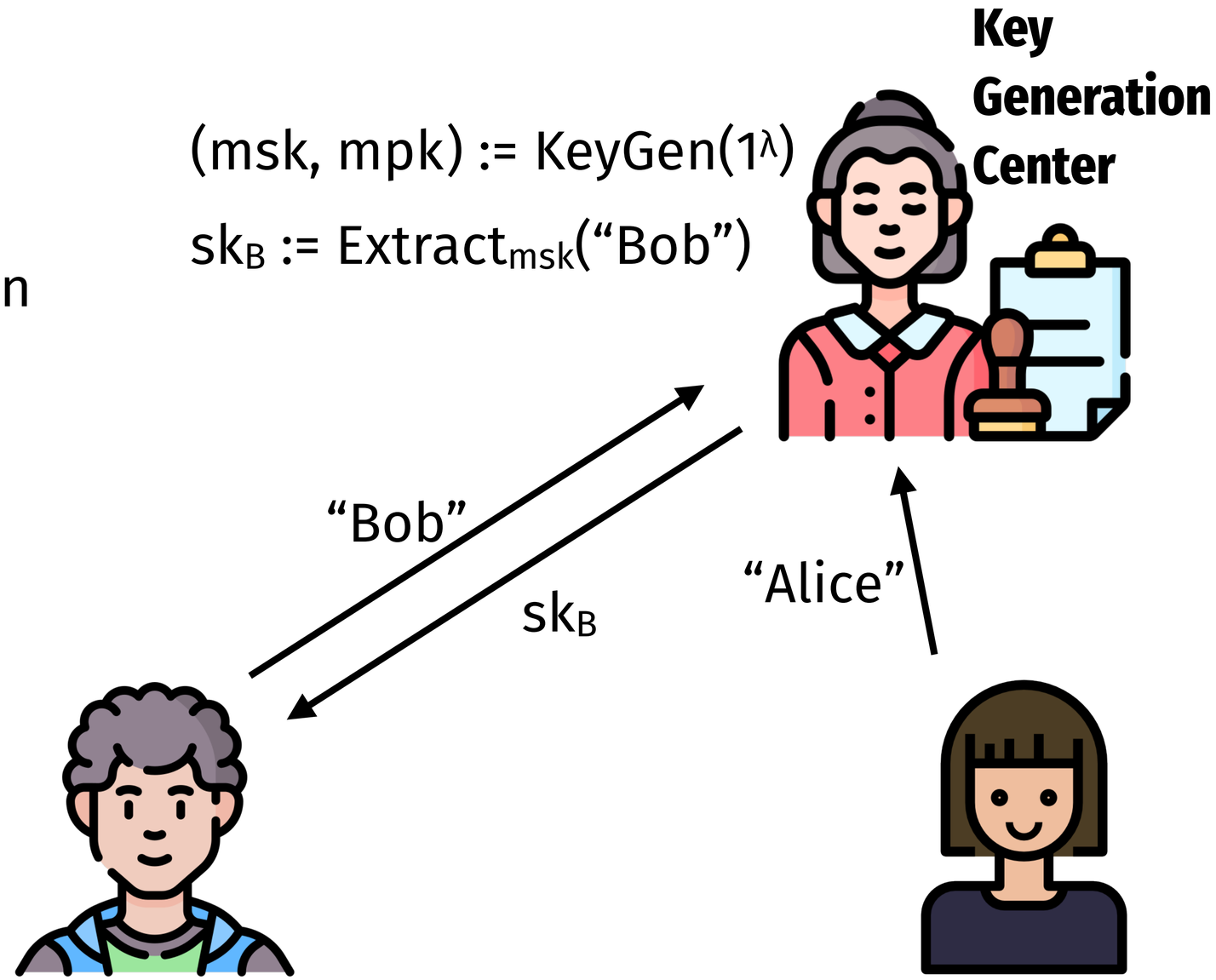
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

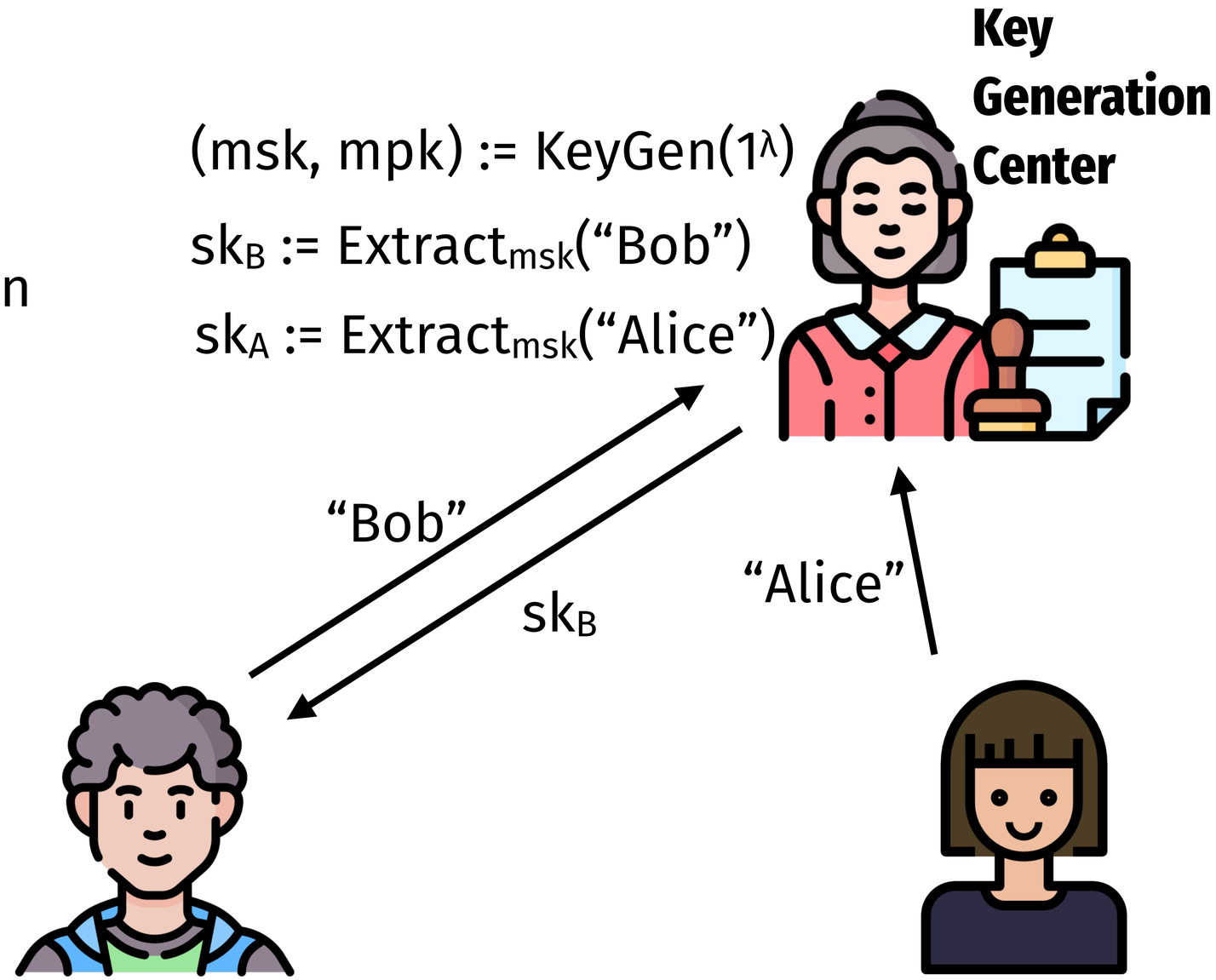
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

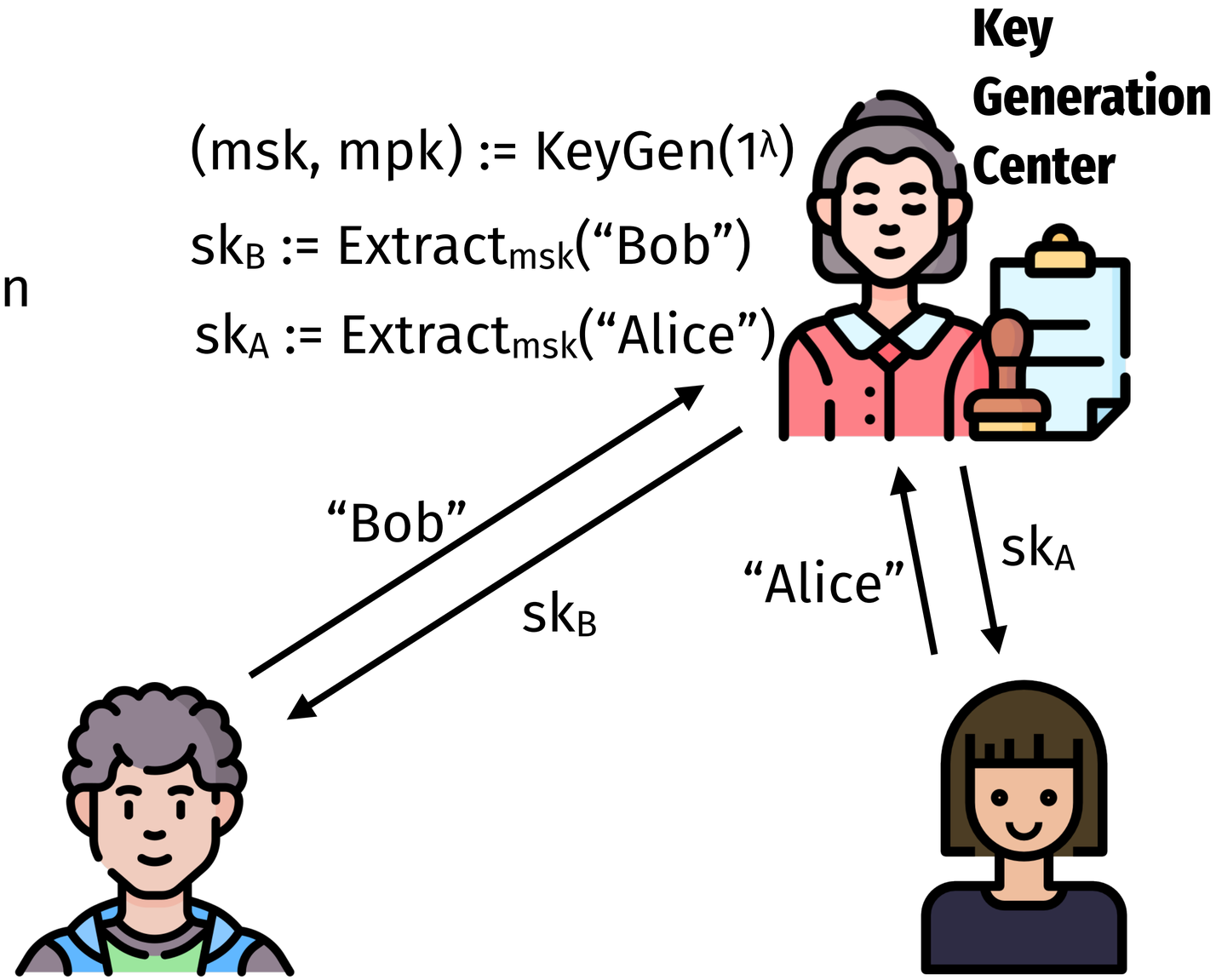
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

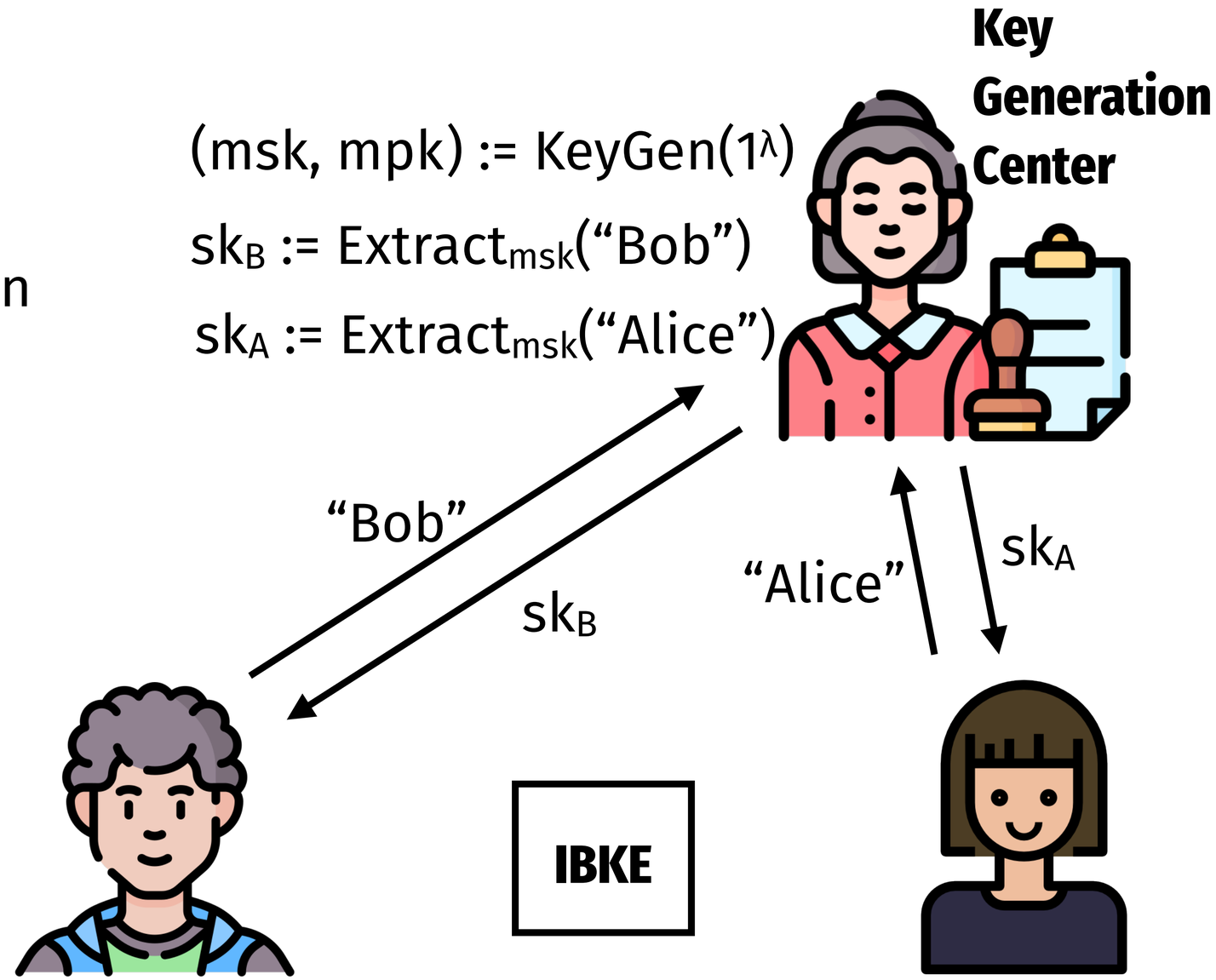
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

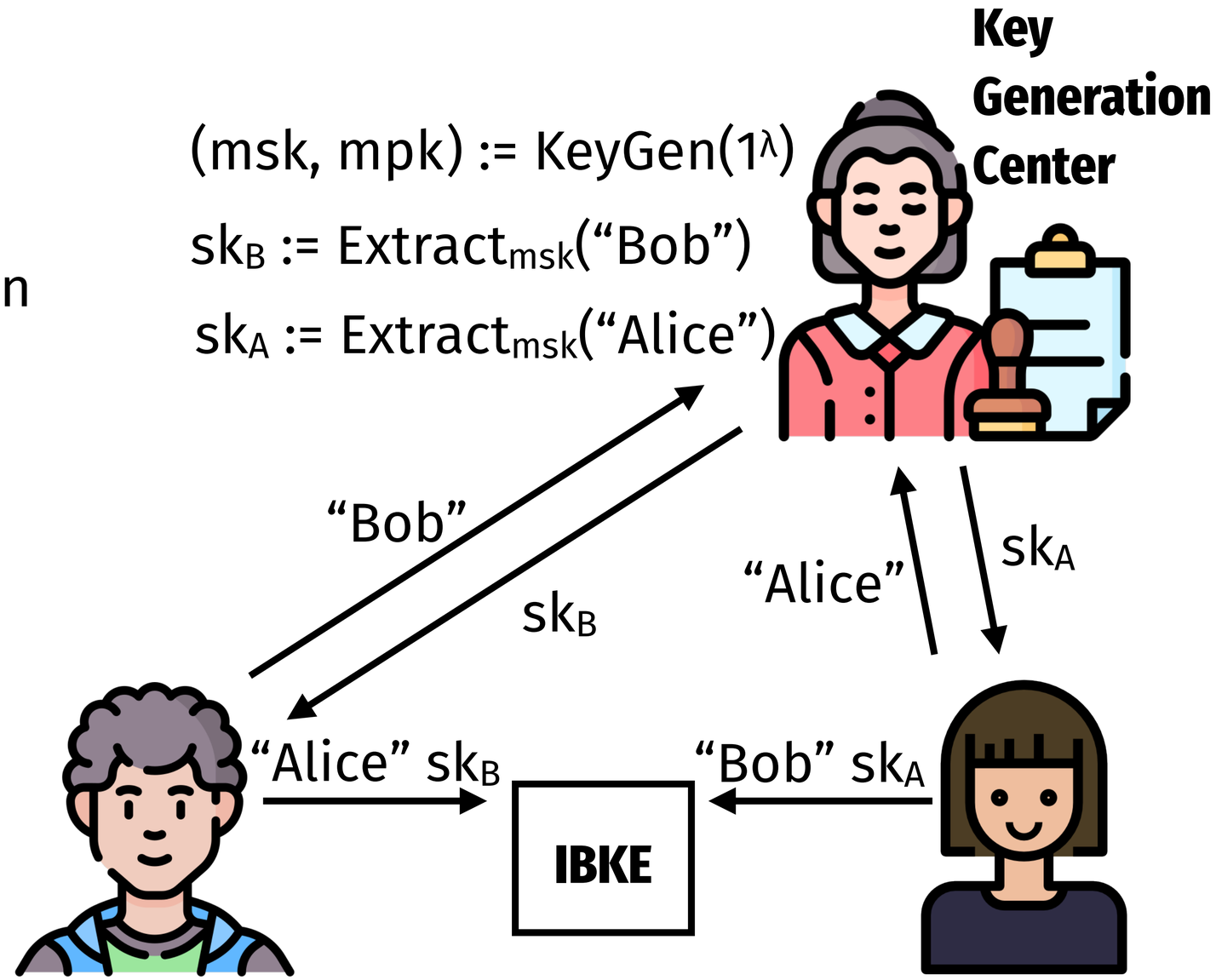
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

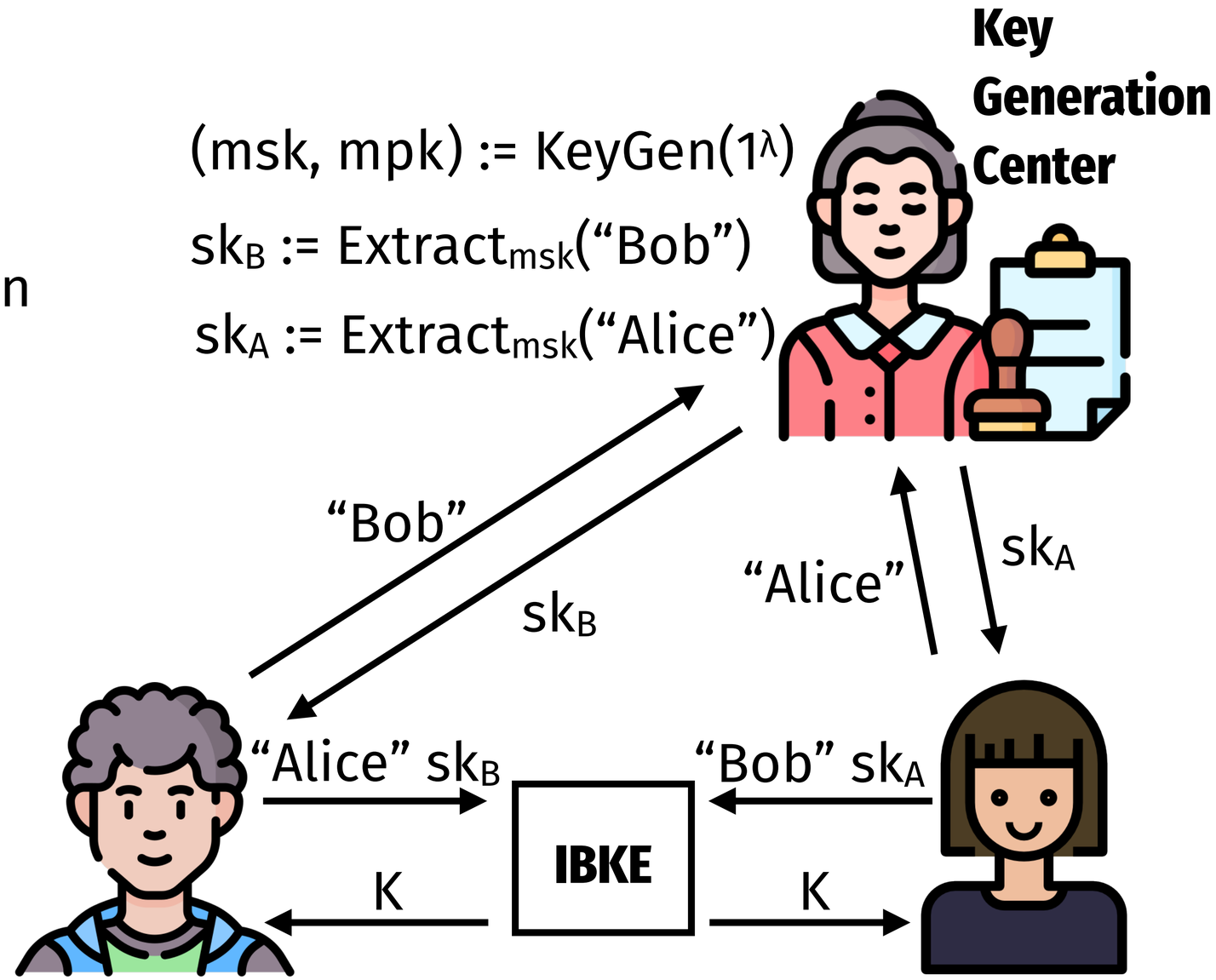
This sounds like identity-based cryptography!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

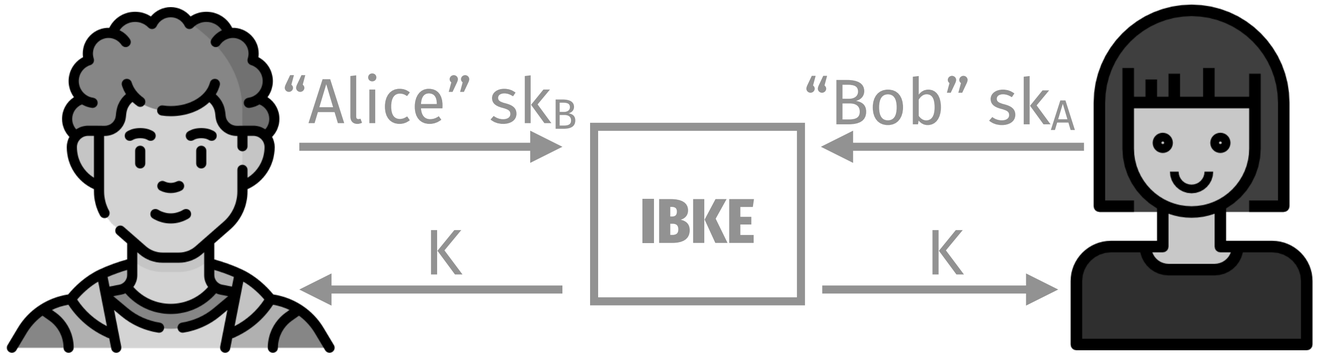


The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:



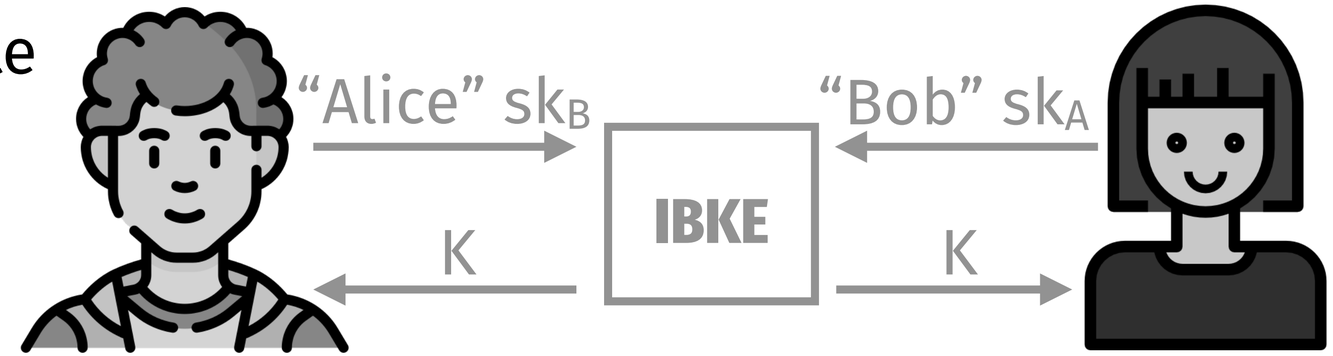
The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

Knowing pw ⇒ can communicate



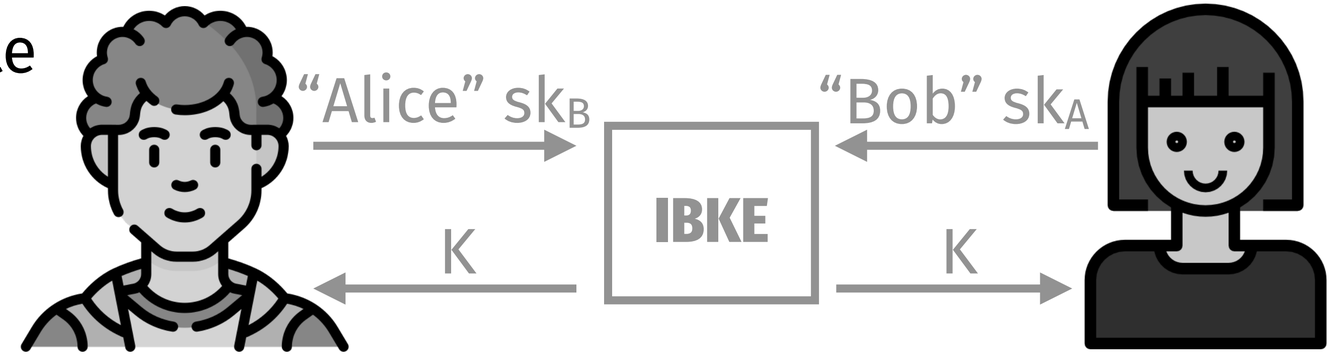
The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile



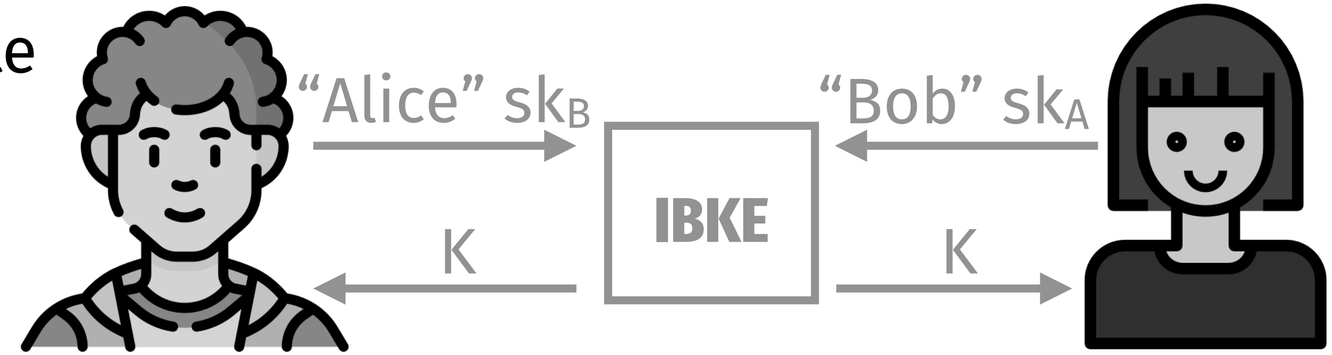
The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

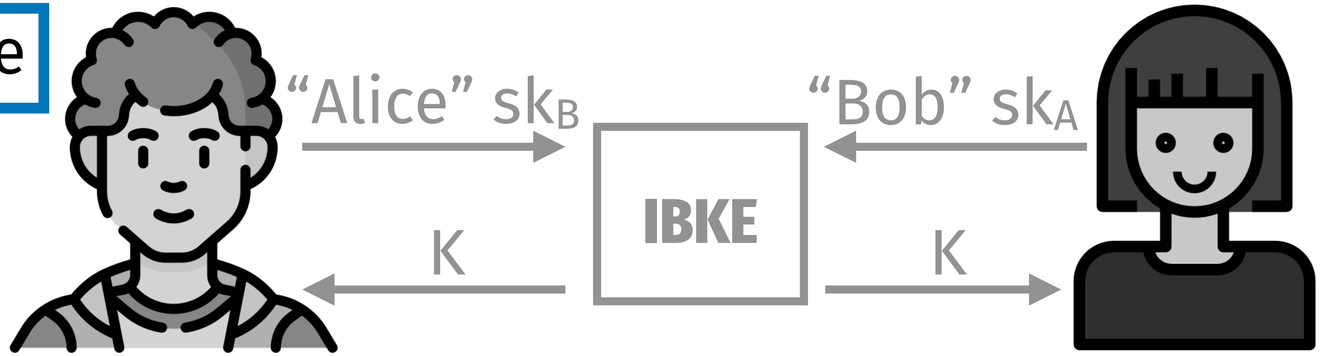
This sounds like identity-based cryptography!

For the password context, need:

Knowing pw ⇒ can communicate

Every ID gets a unique pwfile

Compromise only reveals the pwfile



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

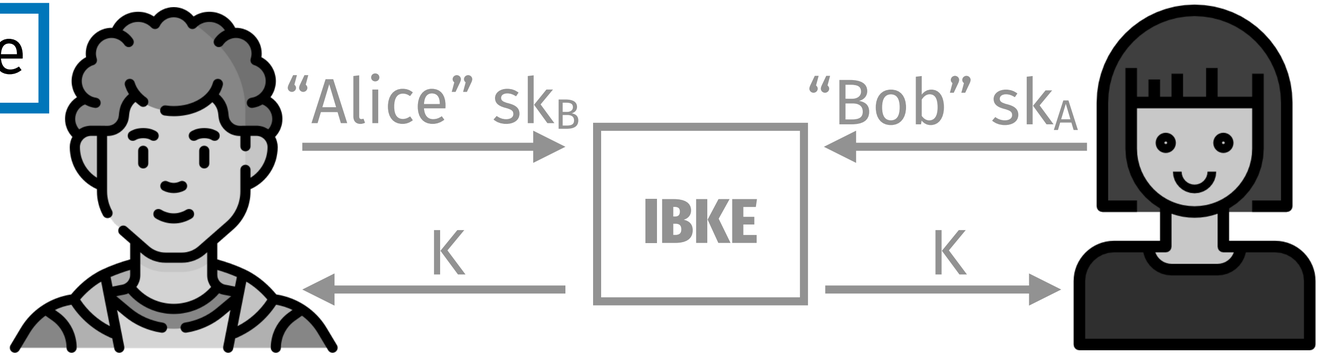
Knowing pw ⇒ can communicate

Every ID gets a unique pwfile

Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

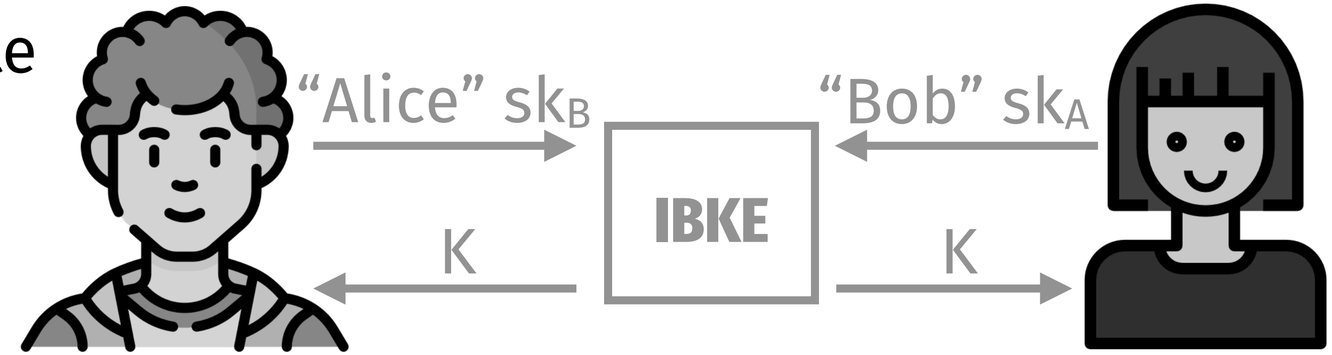
Knowing pw ⇒ can communicate

Every ID gets a unique pwfile

Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

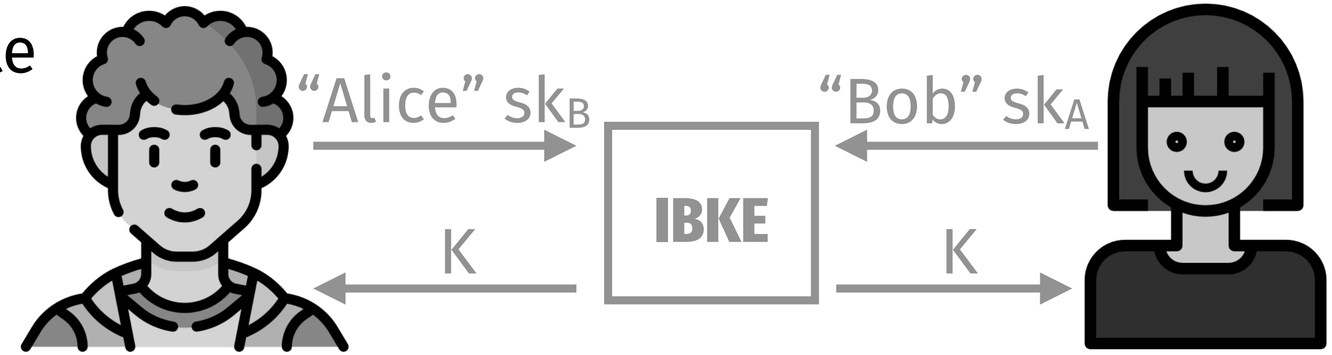
For the password context, need:

- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_B := Extract_{msk}(\text{“Bob”})$



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

Knowing pw ⇒ can communicate

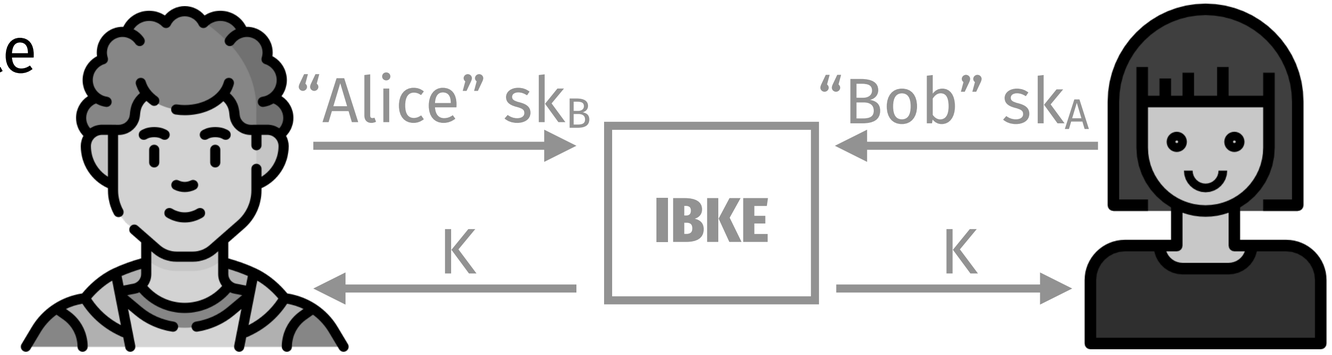
Every ID gets a unique pwfile

Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_B := Extract_{msk}(\text{“Bob”})$



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

Knowing pw ⇒ can communicate

Every ID gets a unique pwfile

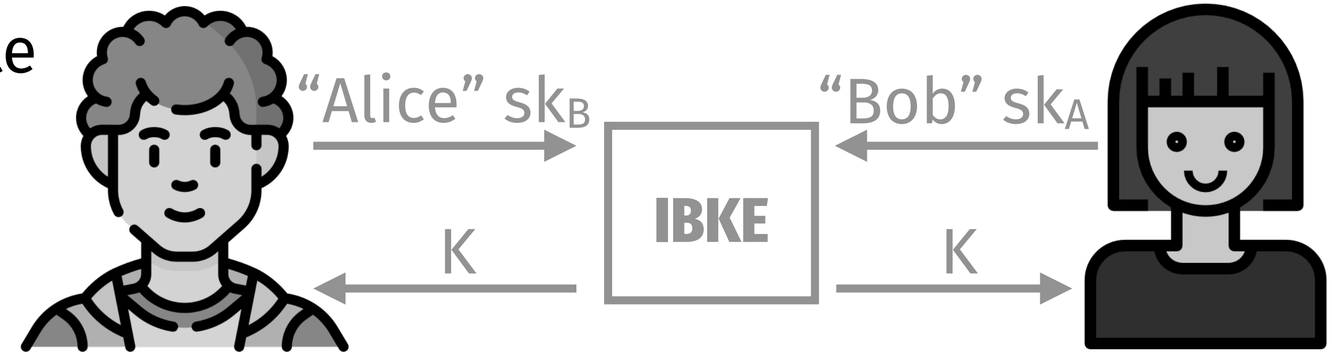
Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_B := Extract_{msk}(\text{“Bob”})$

delete msk and pw



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

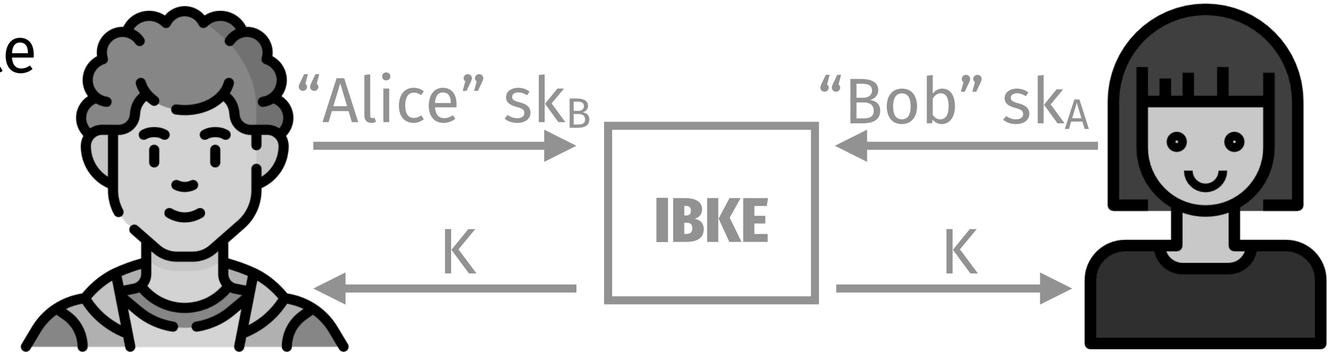
- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_B := Extract_{msk}(\text{“Bob”})$

delete msk and pw



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

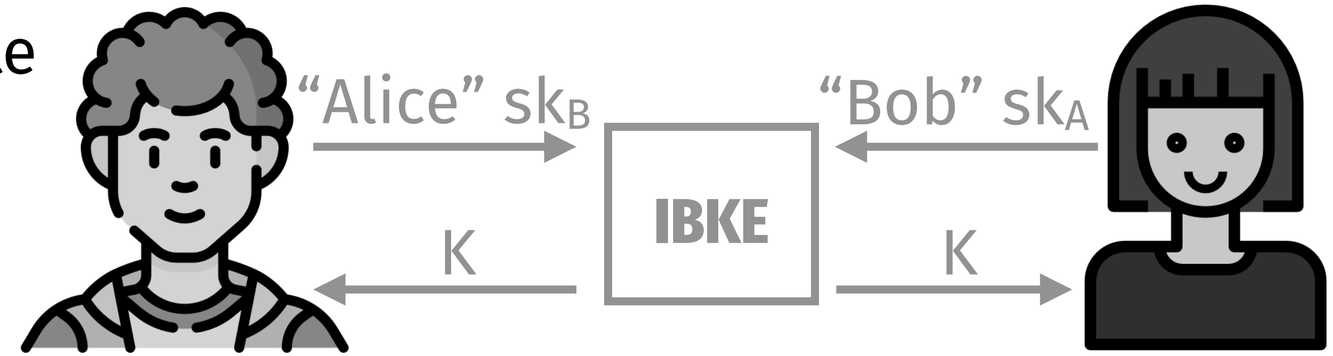
- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile

Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_A := Extract_{msk}(\text{“Alice”})$

delete msk and pw



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile

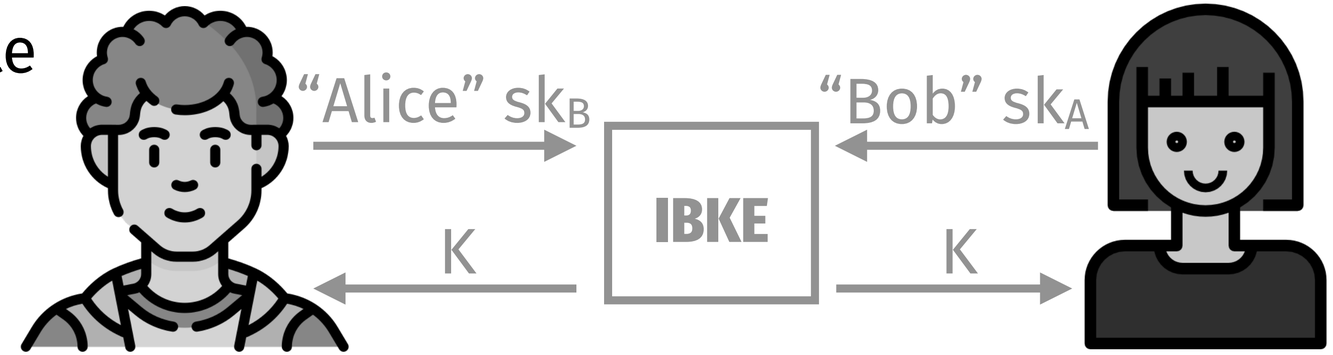
Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_A := Extract_{msk}(\text{“Alice”})$

delete msk and pw

Weird! Identity-based cryptography with no trusted third party (KGC)!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile

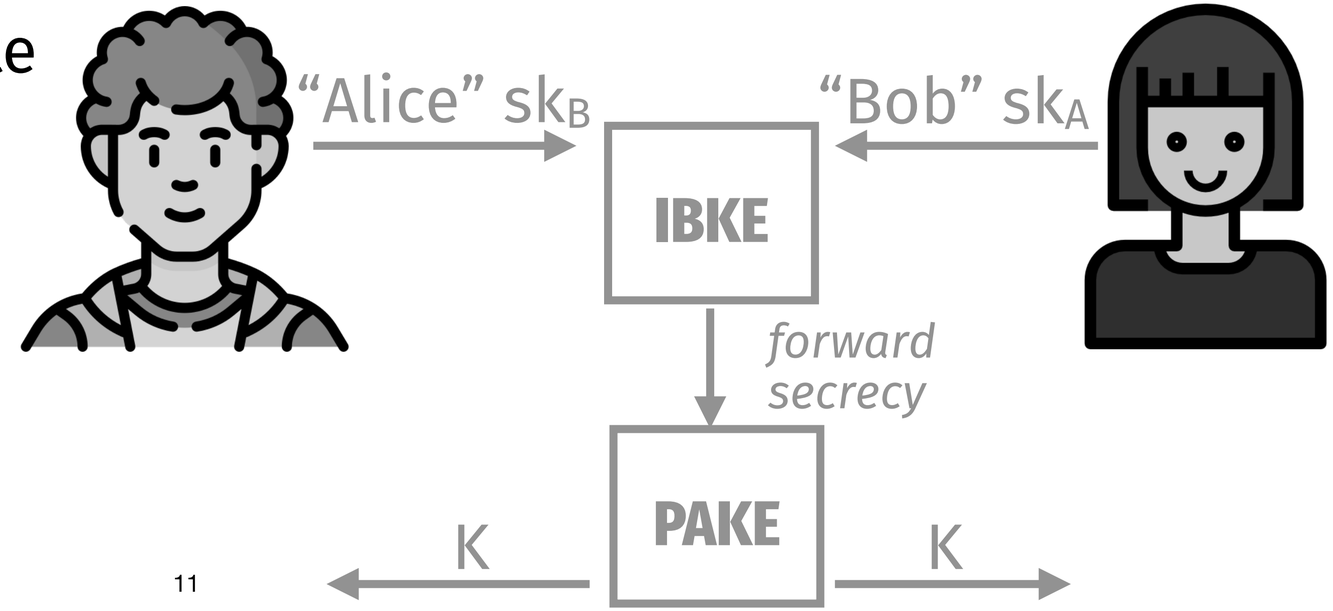
Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_A := Extract_{msk}(\text{“Alice”})$

delete msk and pw

Weird! Identity-based cryptography with no trusted third party (KGC)!



The CHIP iPAKE

Need to bind an ID to a key (pw) in an irreversible way.

This sounds like identity-based cryptography!

For the password context, need:

- Knowing pw ⇒ can communicate
- Every ID gets a unique pwfile
- Compromise only reveals the pwfile

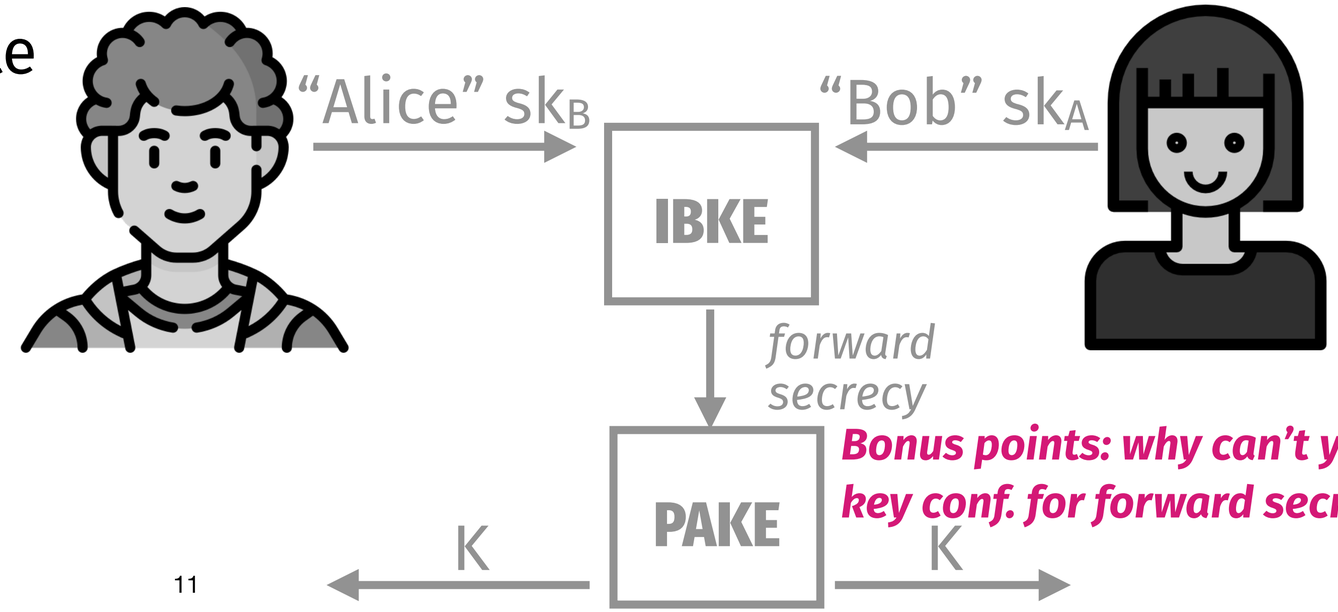
Generating pwfile

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_A := Extract_{msk}(\text{“Alice”})$

delete msk and pw

Weird! Identity-based cryptography with no trusted third party (KGC)!



Bonus points: why can't you use key conf. for forward secrecy?

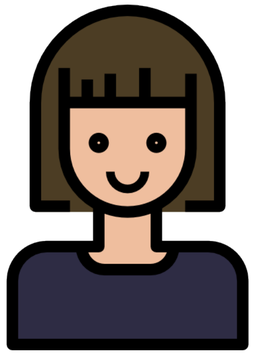
The CHIP iPAKE

The CHIP iPAKE

Problem: This isn’t secure for all IBKE

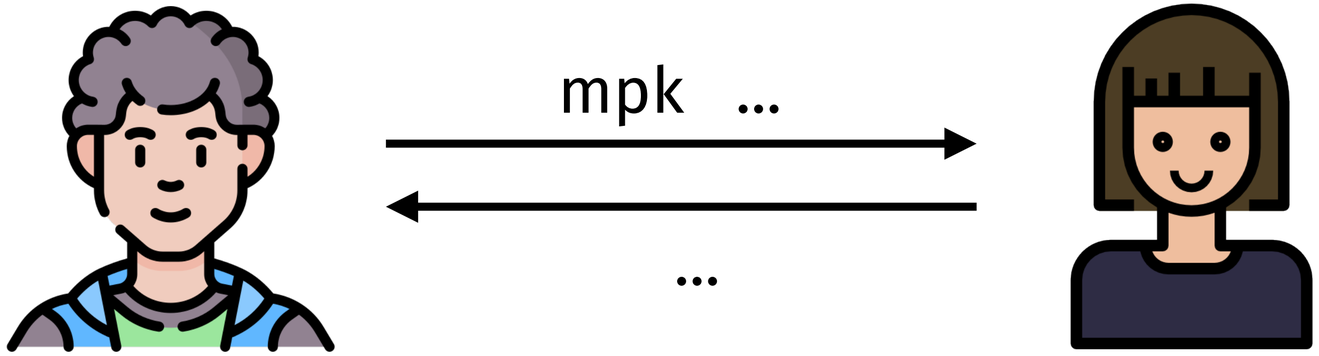
The CHIP iPAKE

Problem: This isn't secure for all IBKE



The CHIP iPAKE

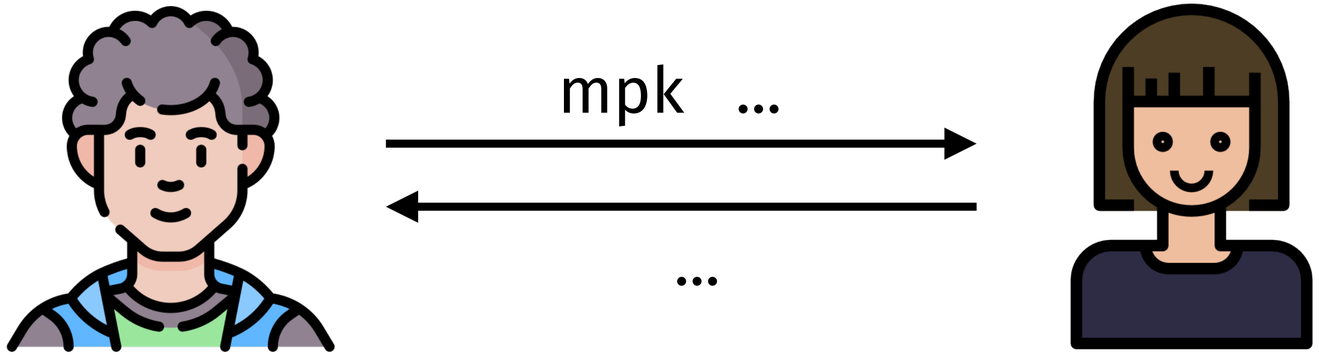
Problem: This isn't secure for all IBKE



The CHIP iPAKE

Problem: This isn't secure for all IBKE

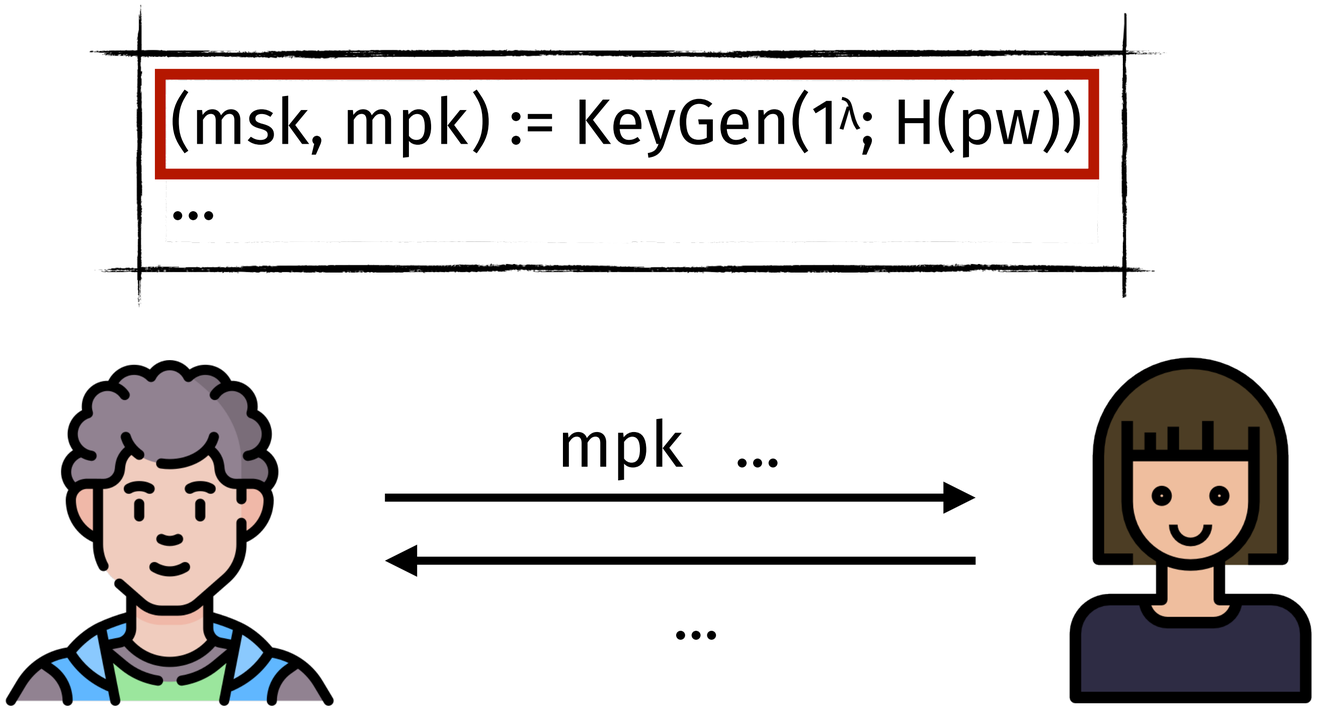
(msk, mpk) := KeyGen(1^λ; H(pw))
...



The CHIP iPAKE

Problem: This isn't secure for all IBKE

mpk leaks data about H(pw)



The CHIP iPAKE

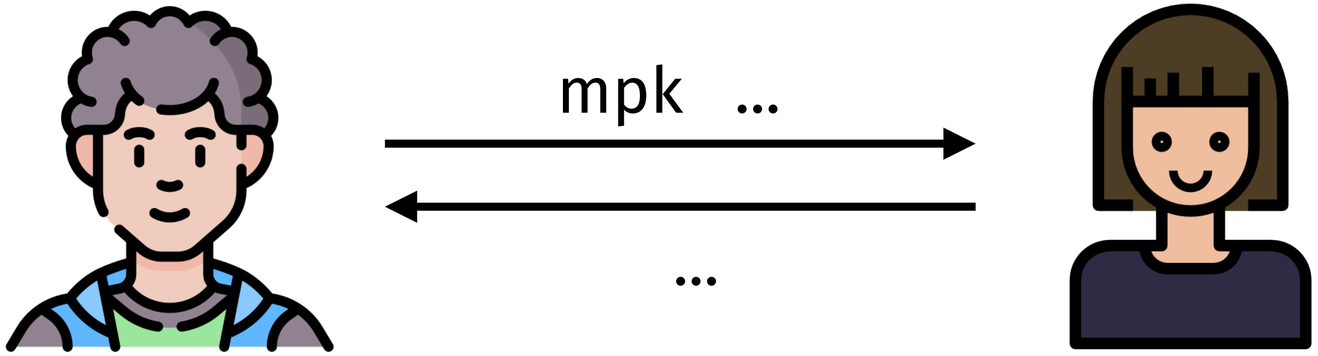
Problem: This isn't secure for all IBKE

mpk leaks data about H(pw)

Passive adversary can brute-force!

$(\mathsf{msk}, \mathsf{mpk}) := \mathsf{KeyGen}(1^\lambda; H(\mathsf{pw}))$

...



The CHIP iPAKE

Problem: This isn't secure for all IBKE

mpk leaks data about H(pw)

In general: IBKE transcript must be **independent** of keygen coins

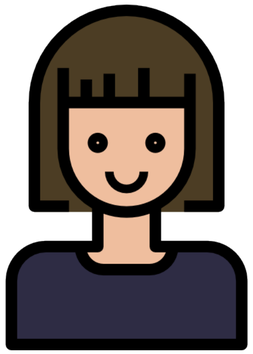
Passive adversary can brute-force!

$(\mathsf{msk}, \mathsf{mpk}) := \mathsf{KeyGen}(1^\lambda; H(\mathsf{pw}))$

...

mpk ...

...



The CHIP iPAKE

Problem: This isn't secure for all IBKE

mpk leaks data about H(pw)

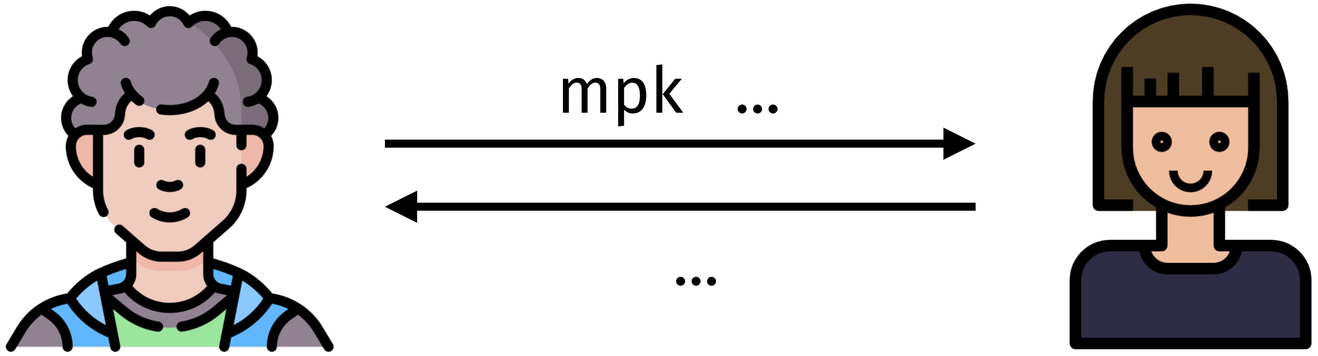
In general: IBKE transcript must be **independent** of keygen coins

No PQ IBKE has this property!

Passive adversary can brute-force!

$(\mathsf{msk}, \mathsf{mpk}) := \mathsf{KeyGen}(1^\lambda; H(\mathsf{pw}))$

...



The CHIP iPAKE

Problem: This isn't secure for all IBKE

mpk leaks data about H(pw)

In general: IBKE transcript must be **independent** of keygen coins

No PQ IBKE has this property!

⇒ No route to make CHIP (or CRISP) post-quantum

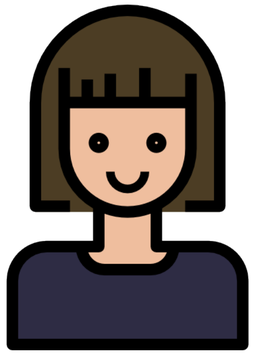
Passive adversary can brute-force!

$(\mathsf{msk}, \mathsf{mpk}) := \mathsf{KeyGen}(1^\lambda; H(\mathsf{pw}))$

...

mpk ...

...



Inspo: The Ω Method aPAKE [GMR06]

Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

- Client has pw
- Server has pwfile (not identity-binding)

Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

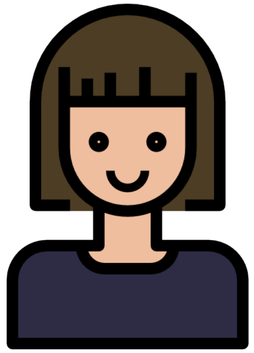
Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

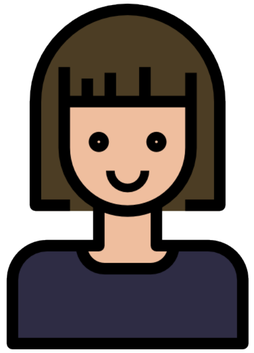
Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

pw



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

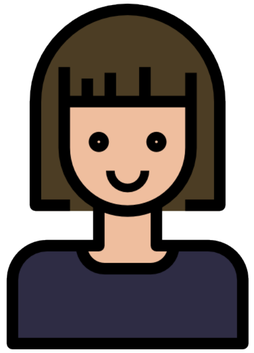
Ω Method — aPAKE from PAKE+signature

Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

pw



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

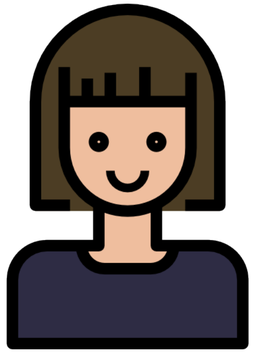
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h2}(sk)$

pw



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Ω.Setup(pw) *exec'd by Alice*

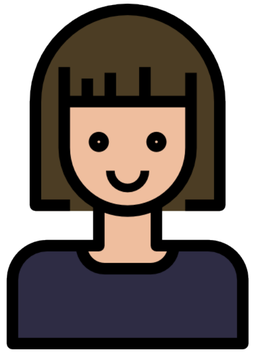
$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2

pw



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

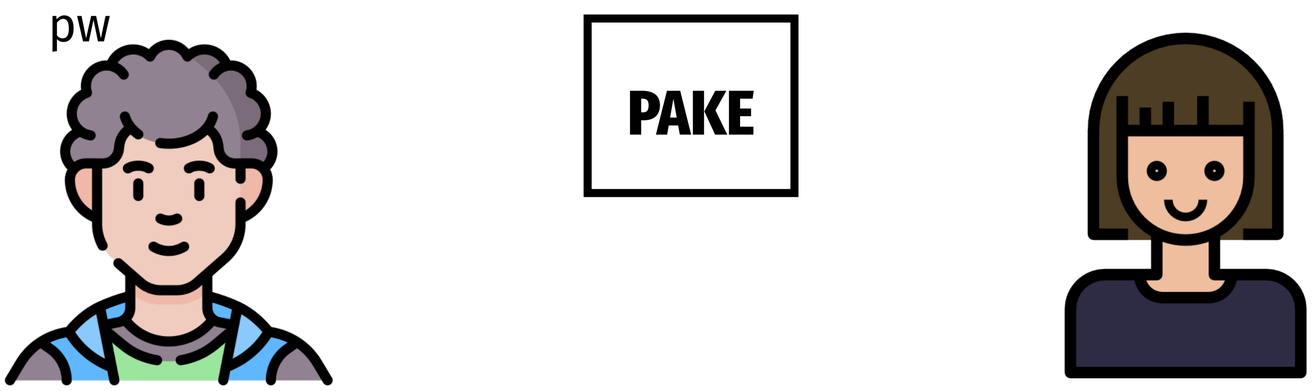
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

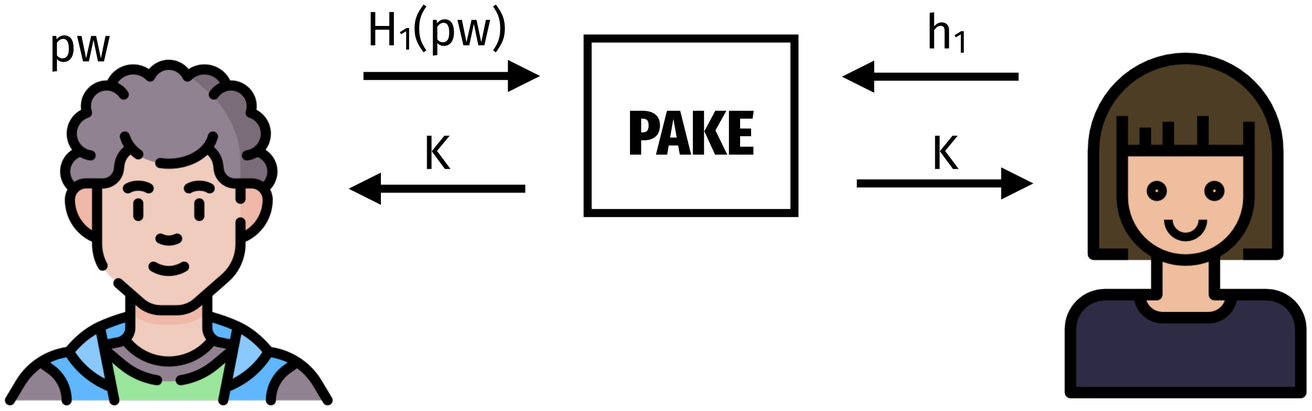
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

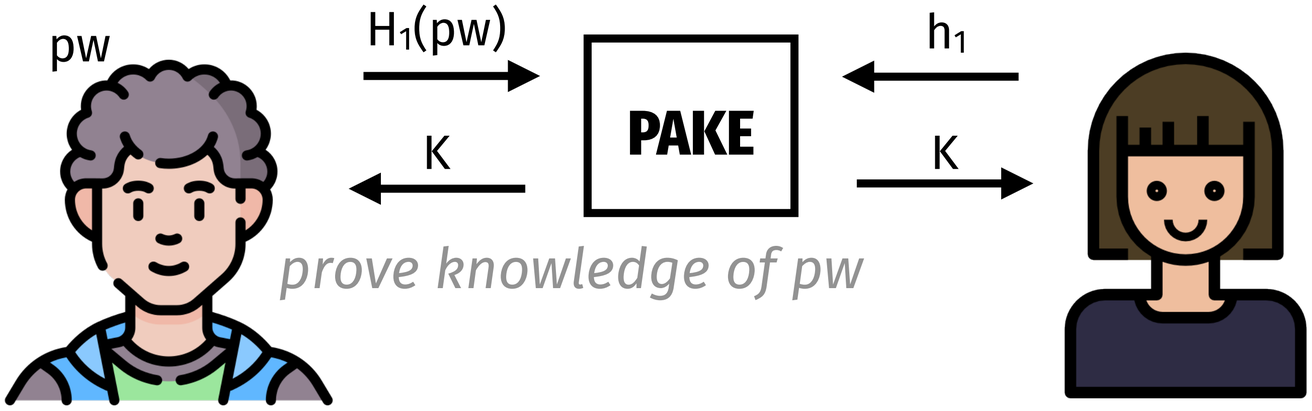
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

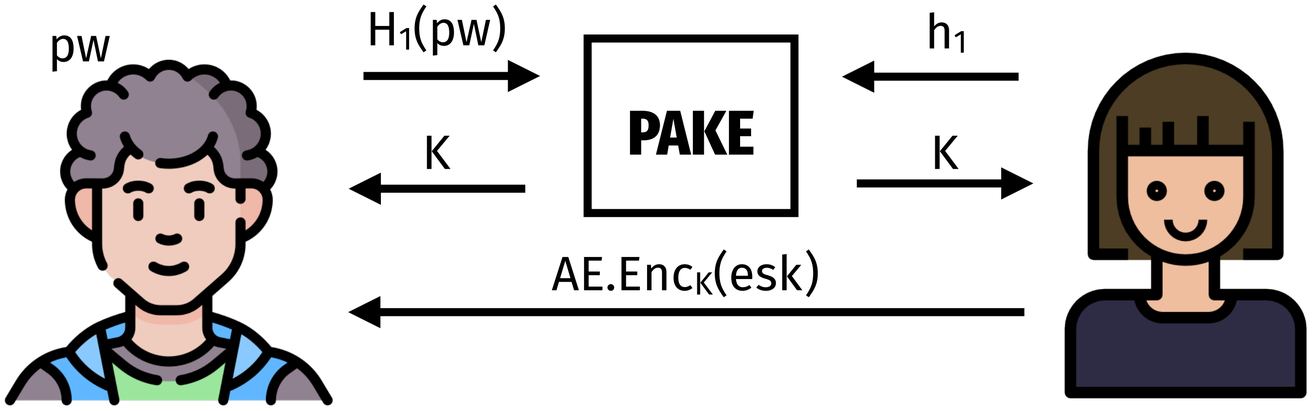
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := \text{Sig.KeyGen}(1^\lambda)$

$esk := \text{AE.Enc}_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

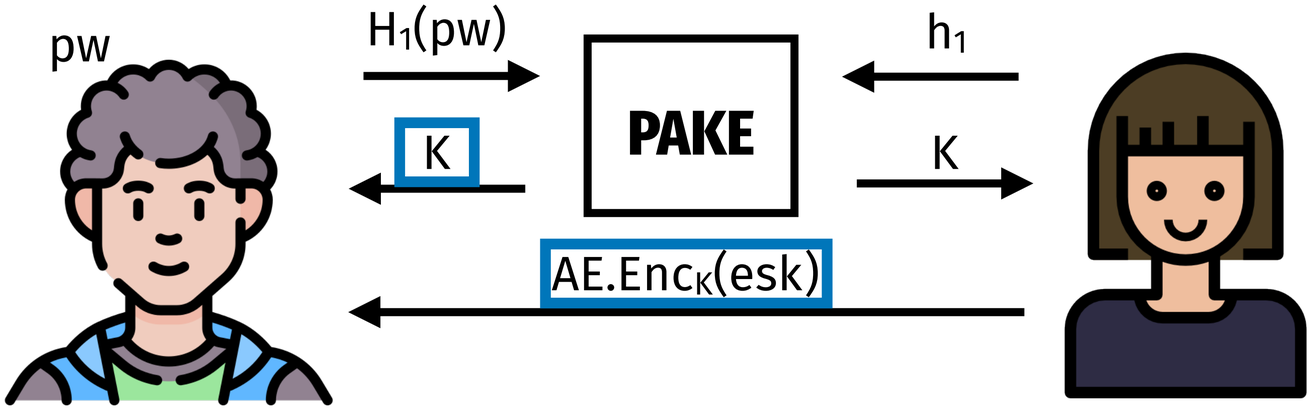
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

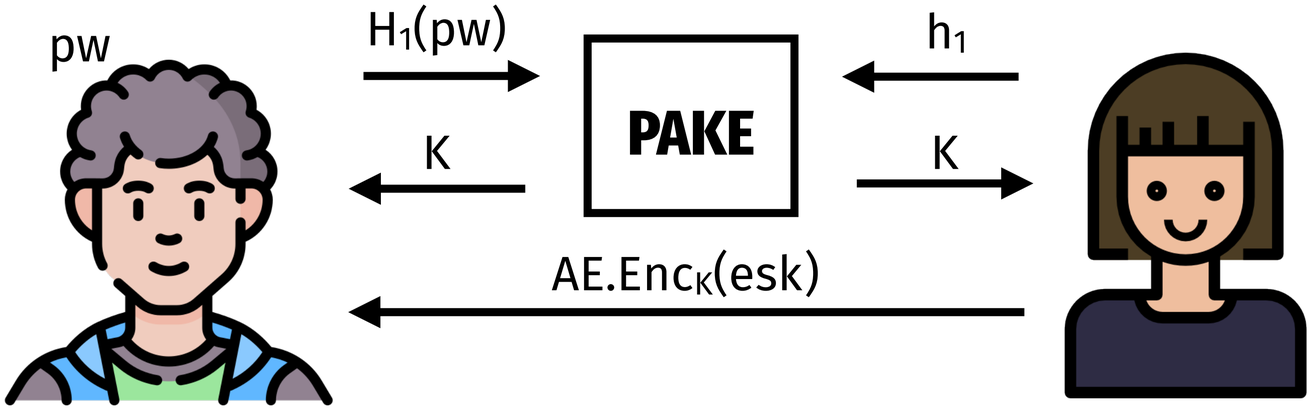
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := \text{Sig.KeyGen}(1^\lambda)$

$esk := \text{AE.Enc}_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

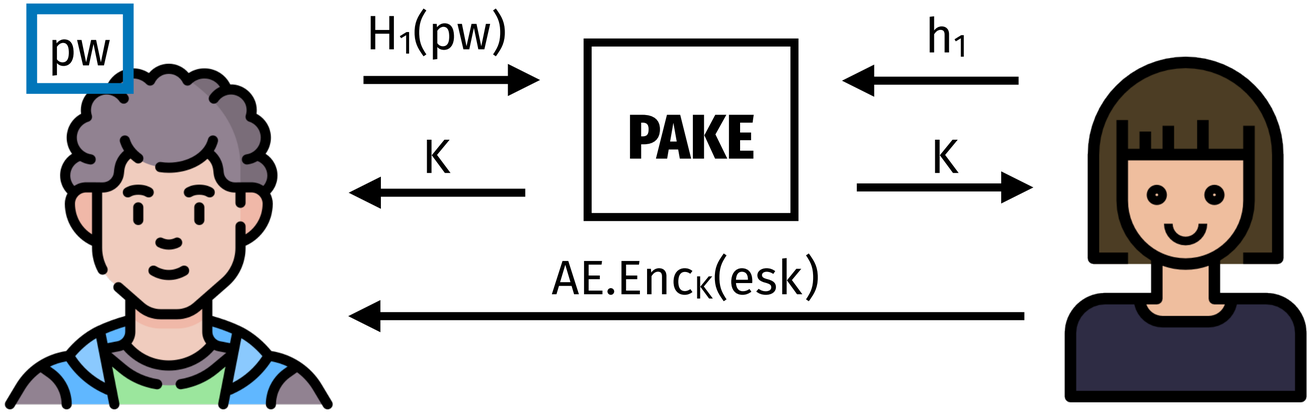
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := \mathrm{Sig.KeyGen}(1^\lambda)$

$esk := \mathrm{AE.Enc}_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

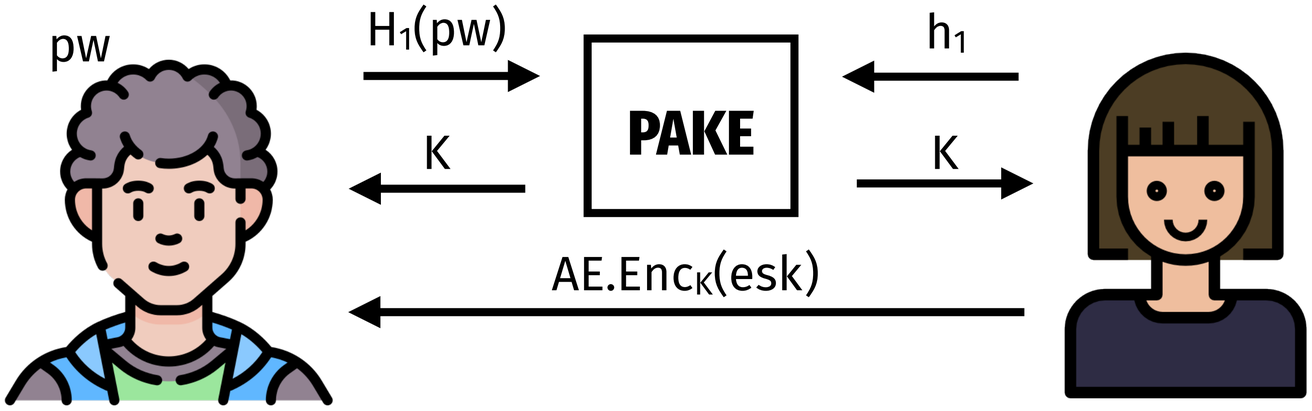
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := \text{Sig.KeyGen}(1^\lambda)$

$esk := \text{AE.Enc}_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

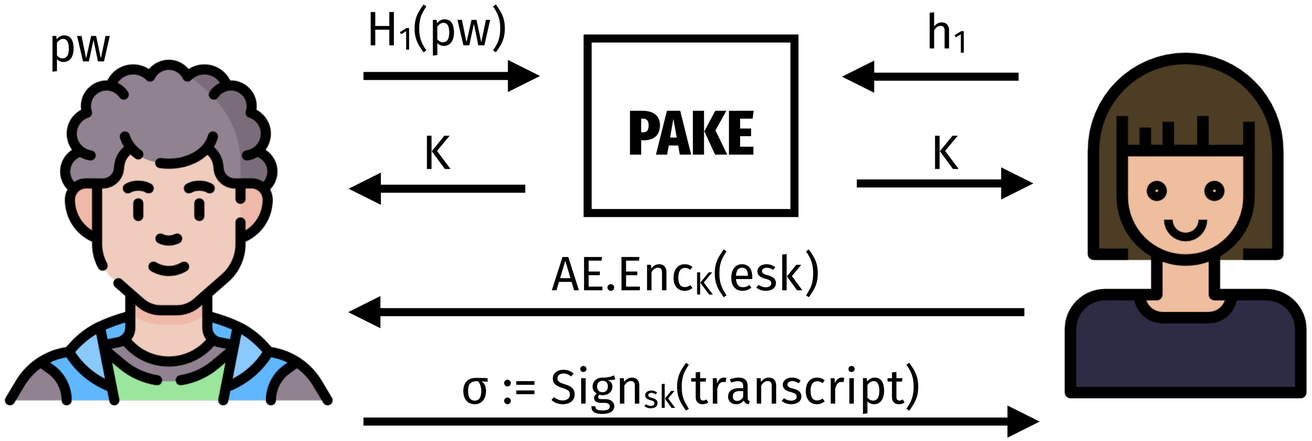
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

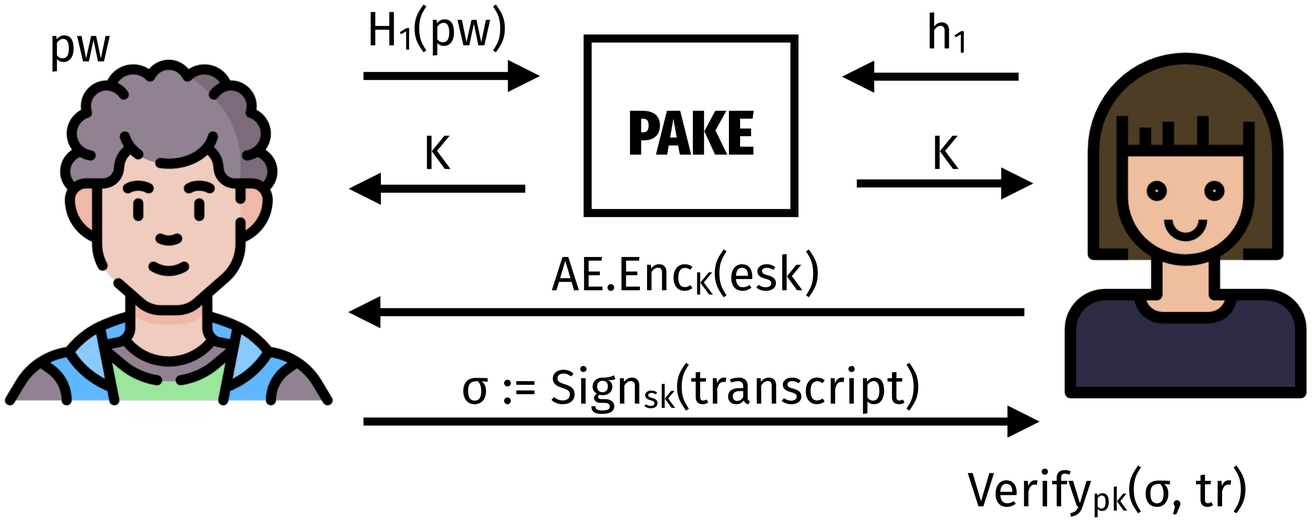
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Observation: **esk leaks data about H_2(pw)**

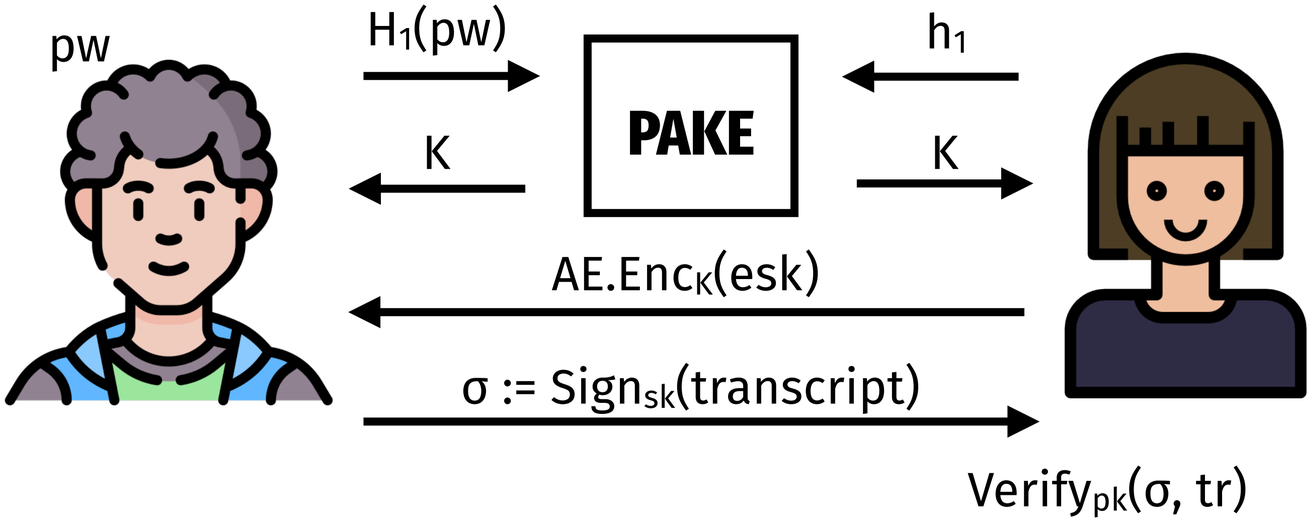
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Observation: **esk leaks data about H_2(pw)**

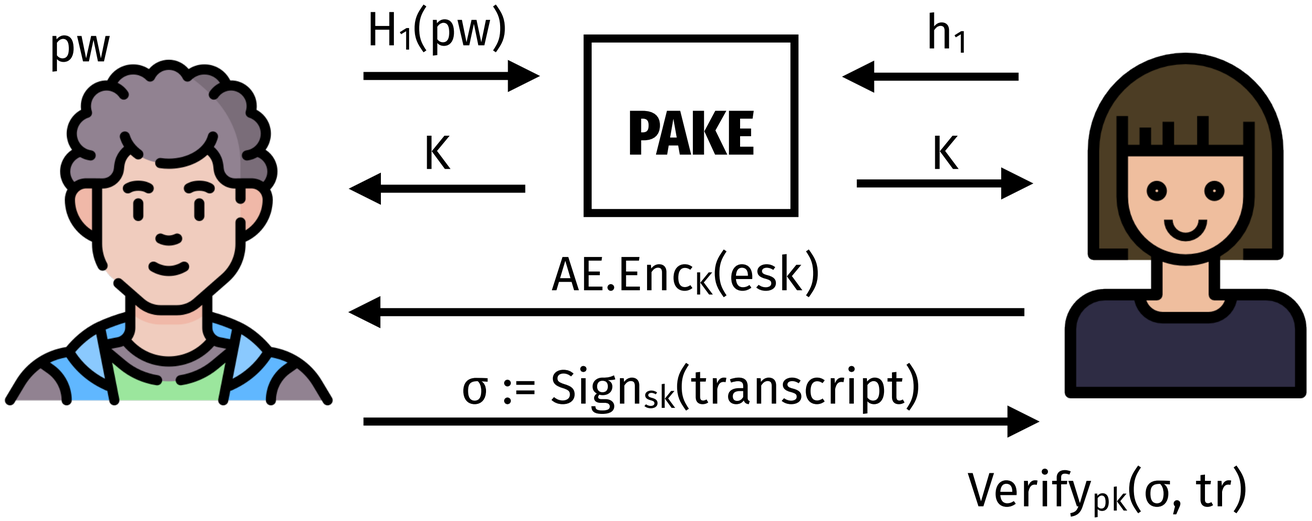
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Observation: **esk leaks data about H_2(pw)**

Encrypting it with K is sufficient, though

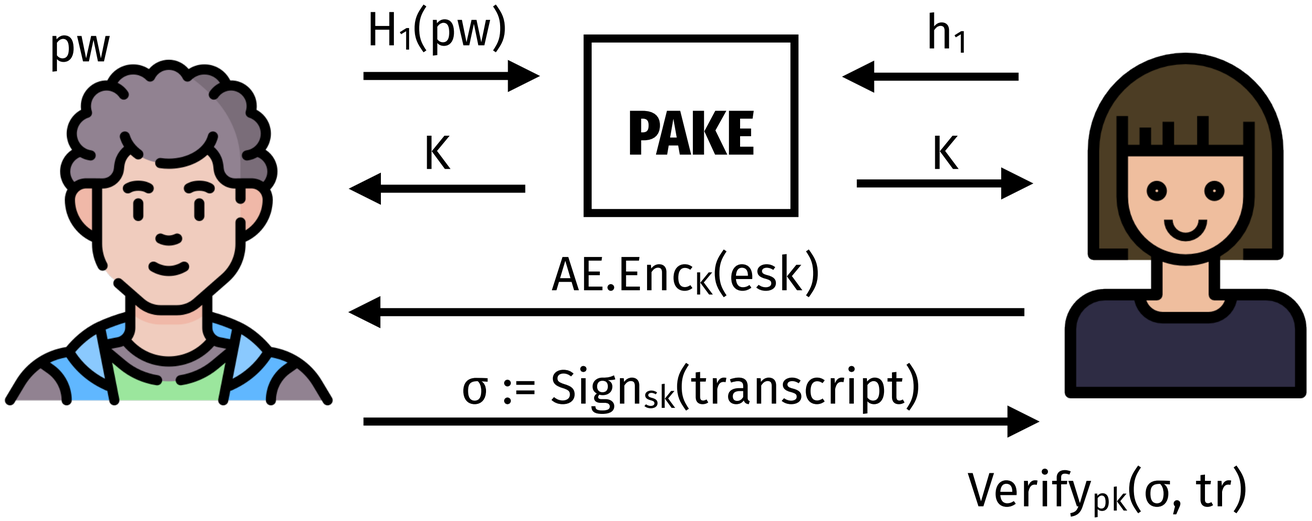
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Observation: **esk leaks data about H_2(pw)**

Encrypting it with K is sufficient, though

If an adversary compromised the PAKE, it knows H_1(pw). So knowing H_2(pw) doesn't help!

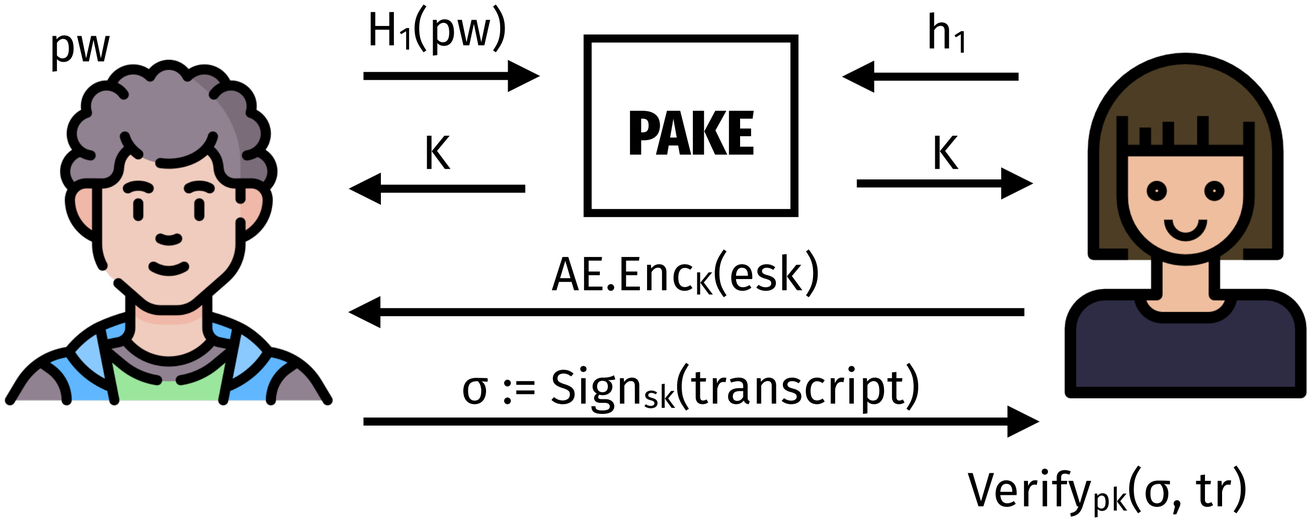
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



Inspo: The Ω Method aPAKE [GMR06]

Augmented PAKE (aPAKE) — PAKE where:

Client has pw

Server has pwfile (not identity-binding)

Ω Method — aPAKE from PAKE+signature

Observation: **esk leaks data about H_2(pw)**

Encrypting it with K is sufficient, though

If an adversary compromised the PAKE, it knows H_1(pw). So knowing H_2(pw) doesn't help!

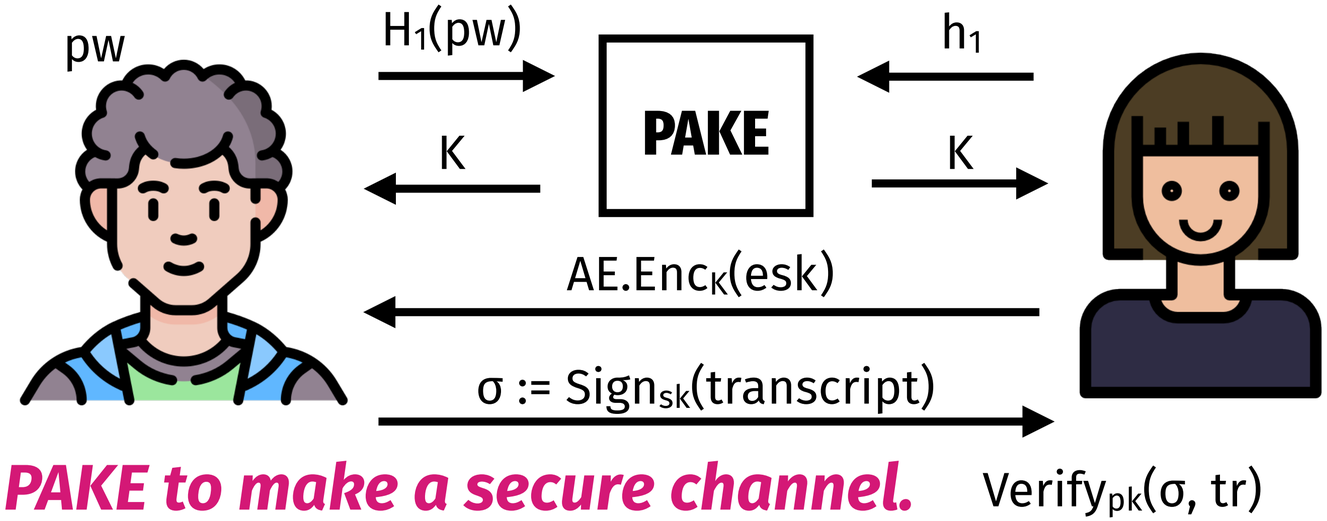
Ω.Setup(pw) *exec'd by Alice*

$(h_1, h_2) := (H_1(pw), H_2(pw))$

$(sk, pk) := Sig.KeyGen(1^\lambda)$

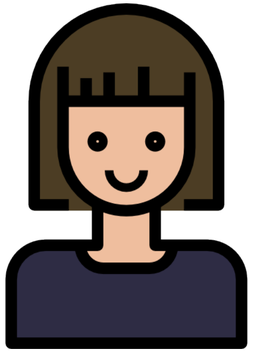
$esk := AE.Enc_{h_2}(sk)$

delete sk, pw, h_2



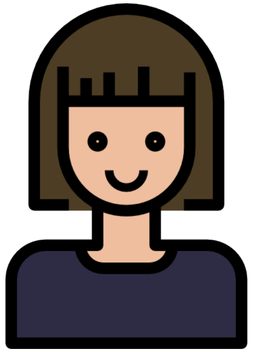
Takeaway: use a PAKE to make a secure channel. Then put leaky protocols in that channel

LATKE Construction



LATKE Construction

Use **self-KGC trick** of CHIP



LATKE Construction

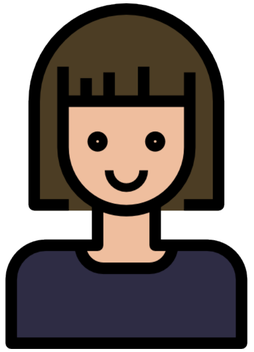
Use **self-KGC trick** of CHIP

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

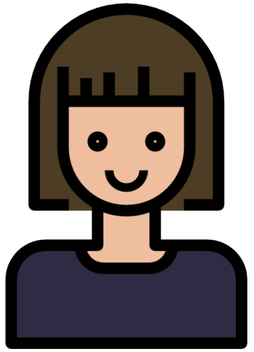
Establish a secure channel like Ω method

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

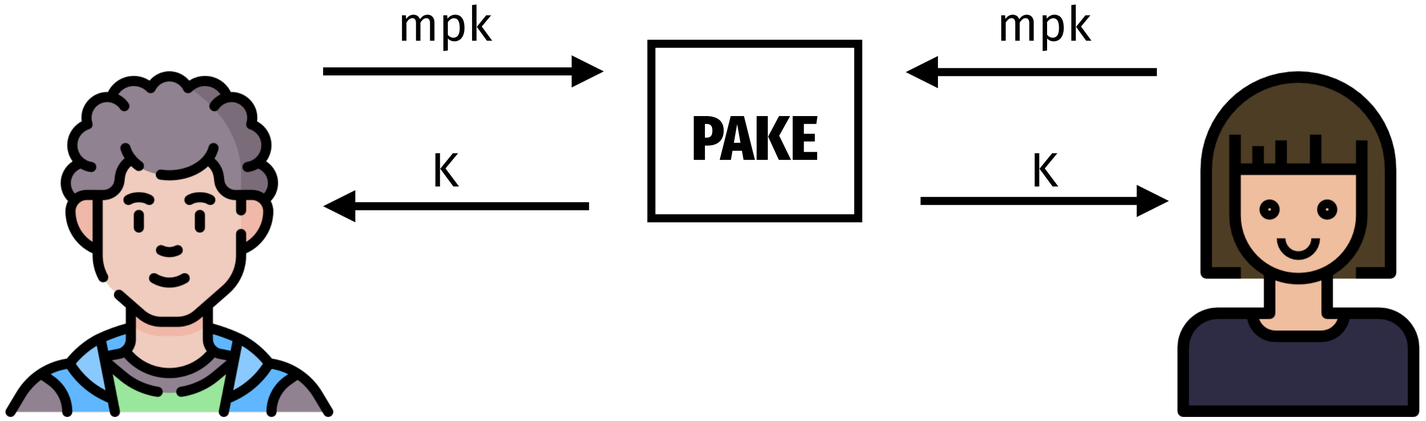
Establish a secure channel like Ω method

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

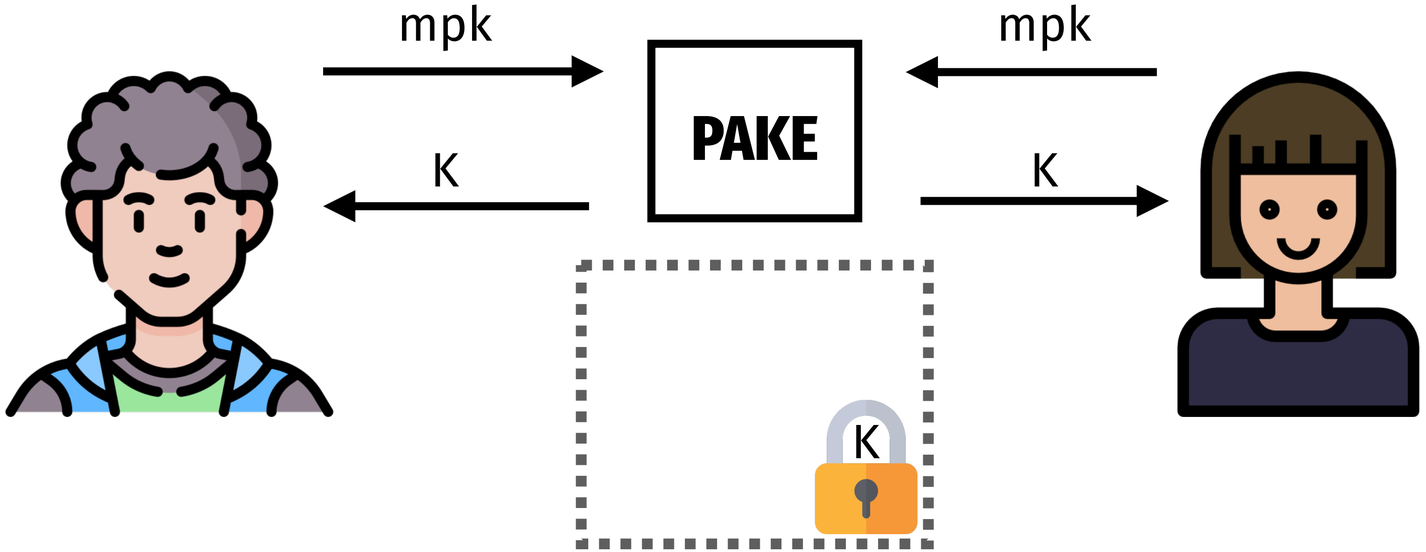
Establish a secure channel like Ω method

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

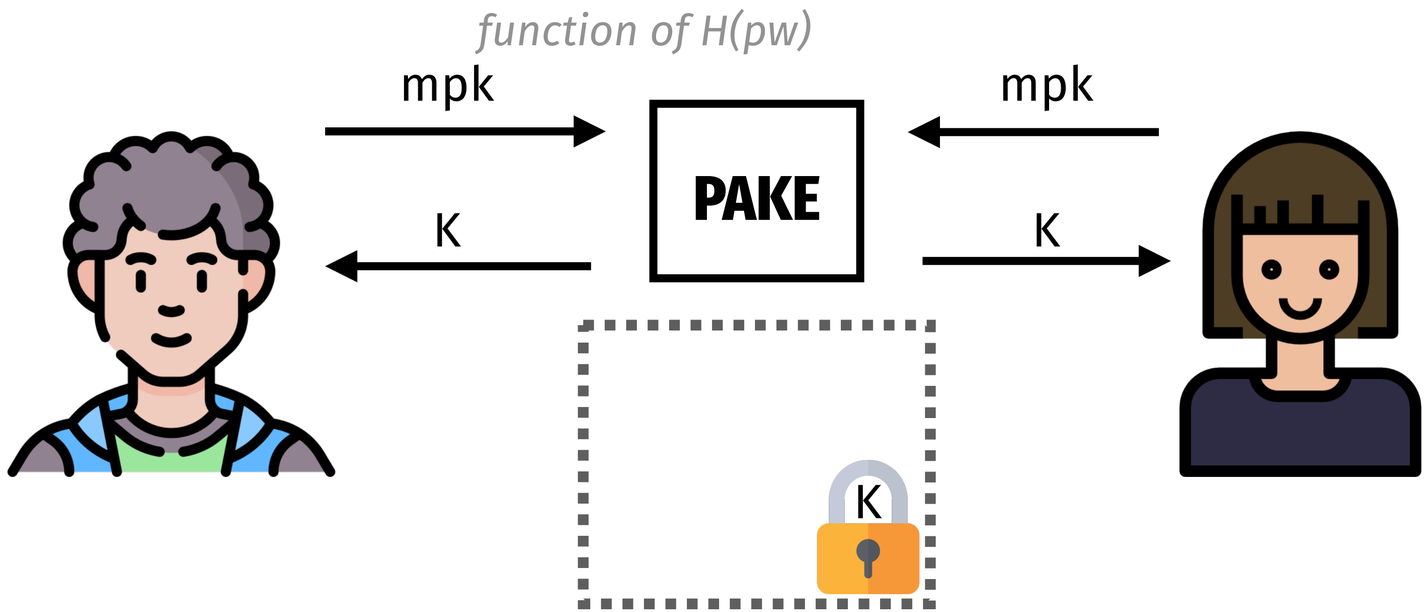
Establish a secure channel like Ω method

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

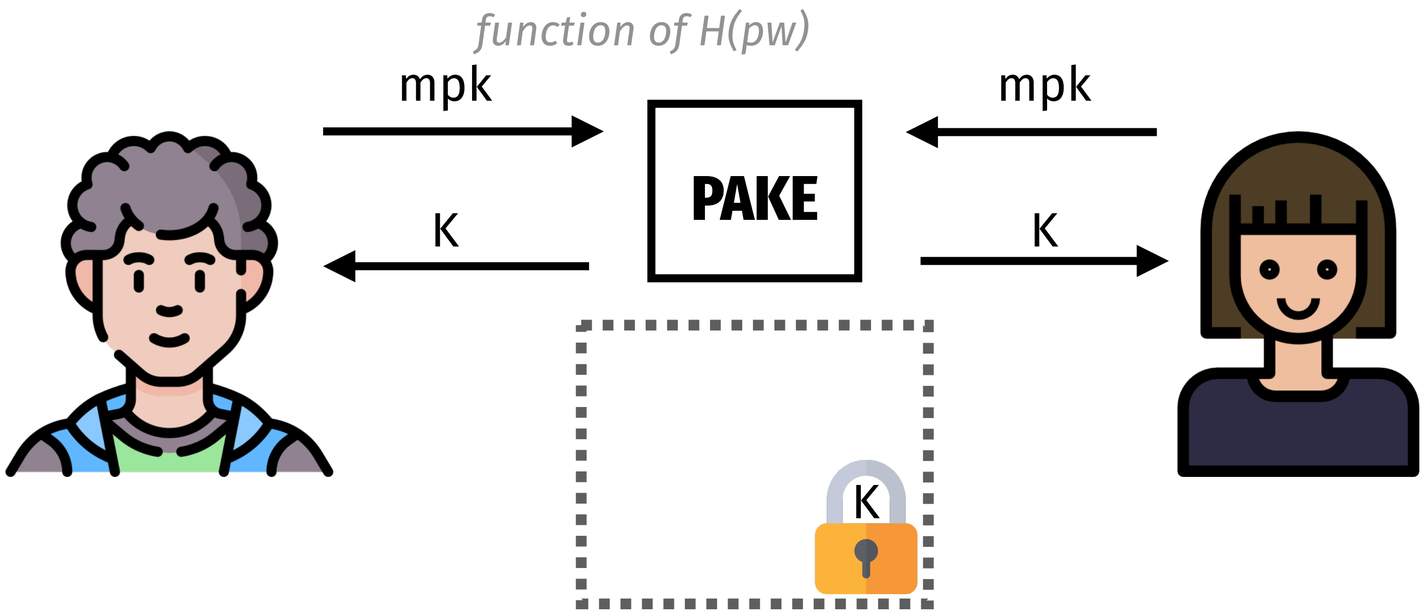
Run **any IBKE in the secure channel**

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

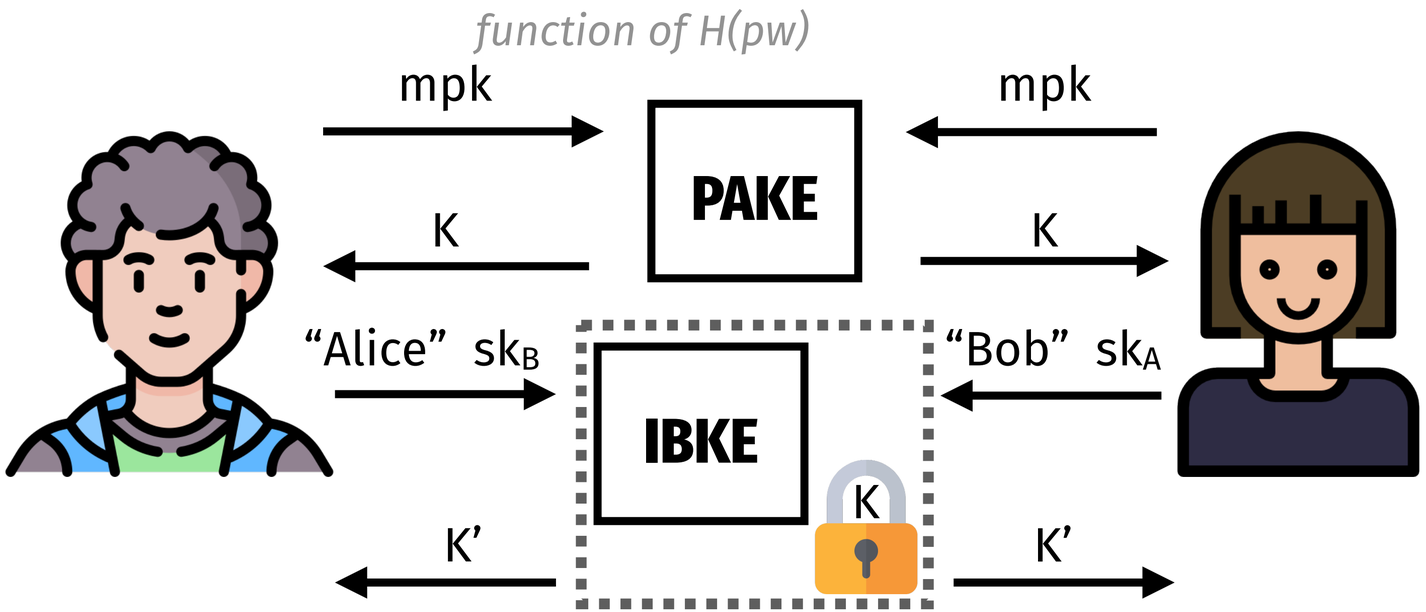
Run **any IBKE in the secure channel**

LATKE.Setup(pw, id) *same as CHIP*

(msk, mpk) := KeyGen(1^λ; H(pw))

sk_{id} := $Extract_{msk}$(id)

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

Run **any IBKE in the secure channel**

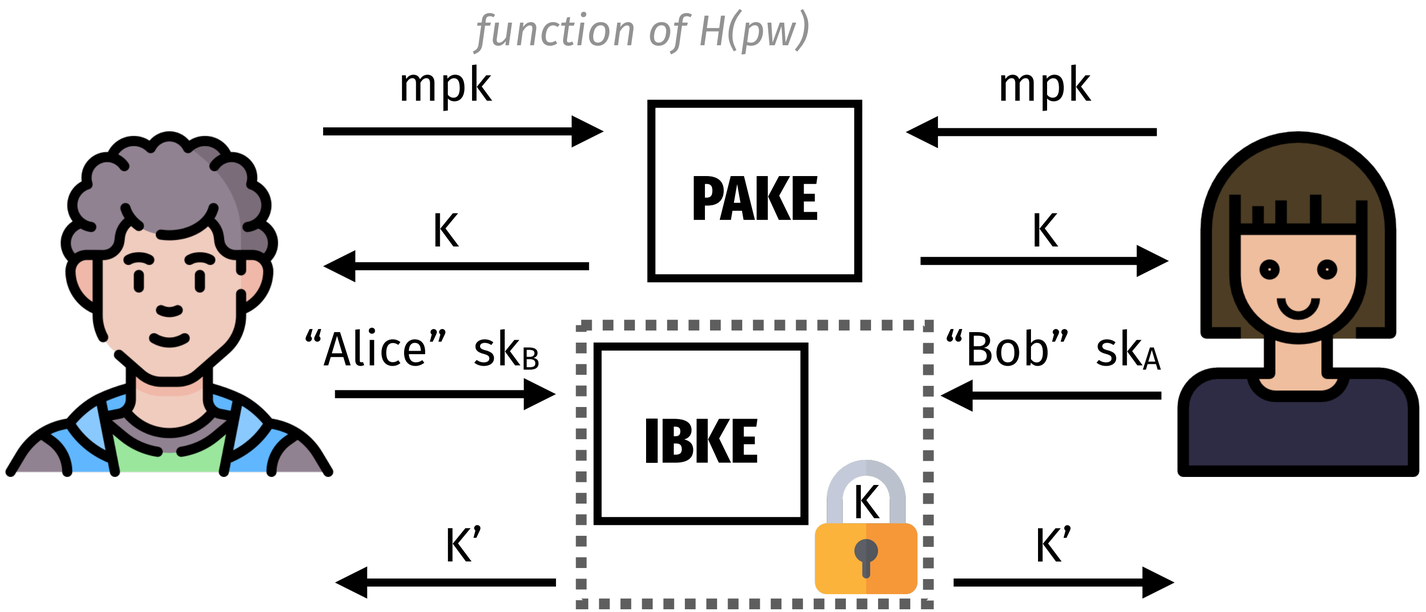
Important details:

LATKE.Setup(pw, id) *same as CHIP*

(msk, mpk) := KeyGen(1^λ; H(pw))

sk_{id} := $Extract_{msk}$(id)

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

Run **any IBKE in the secure channel**

Important details:

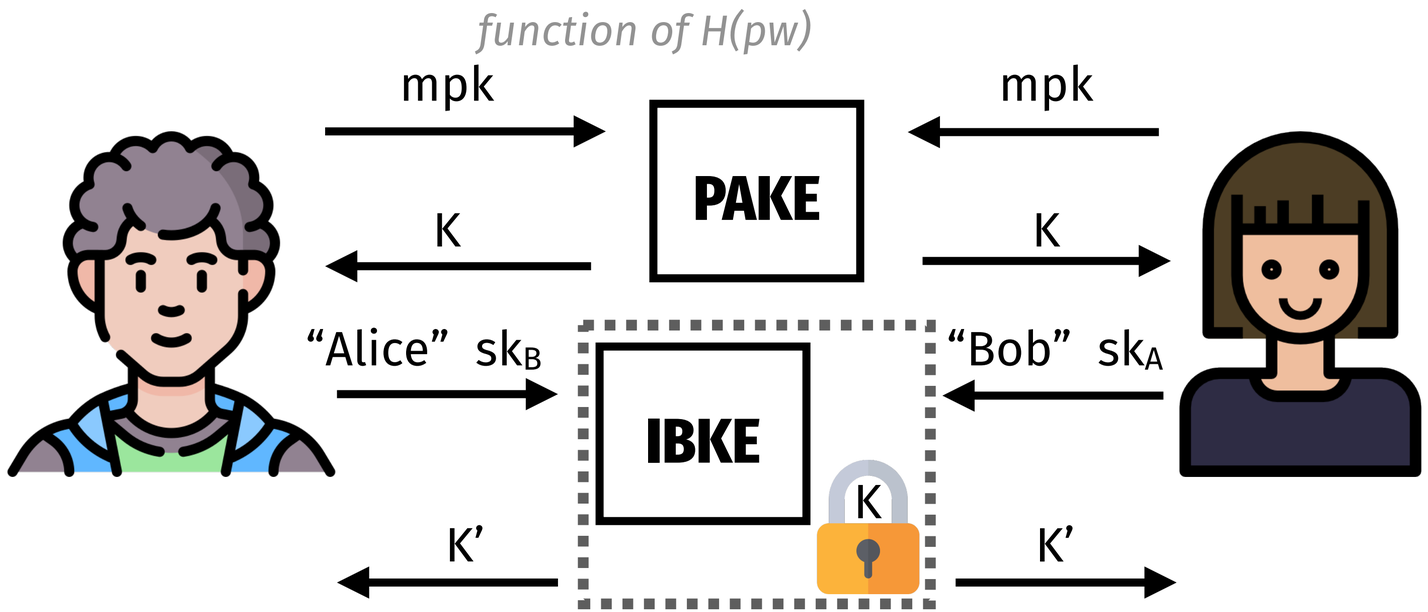
"Secure channel" (EUE transform)

LATKE.Setup(pw, id) *same as CHIP*

(msk, mpk) := KeyGen(1^λ; H(pw))

sk_{id} := $Extract_{msk}$(id)

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

Run **any IBKE in the secure channel**

Important details:

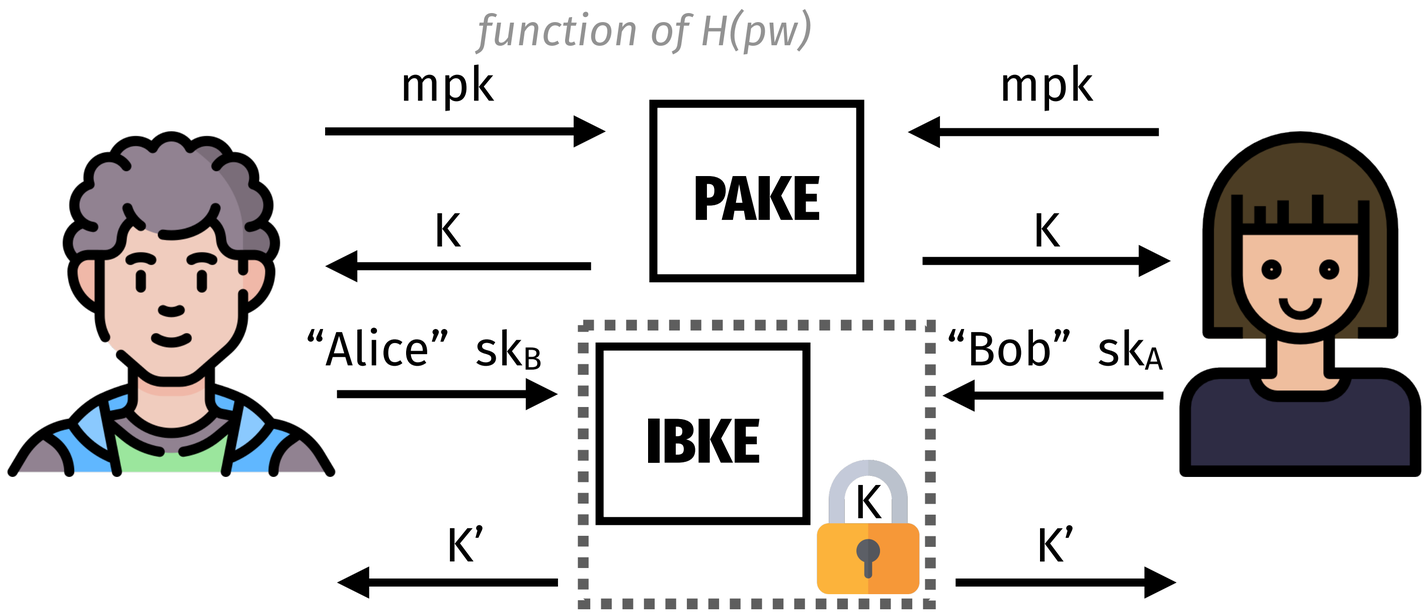
- "Secure channel" (EUE transform)
- How to make a PQ IBKE

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

Run **any IBKE in the secure channel**

Important details:

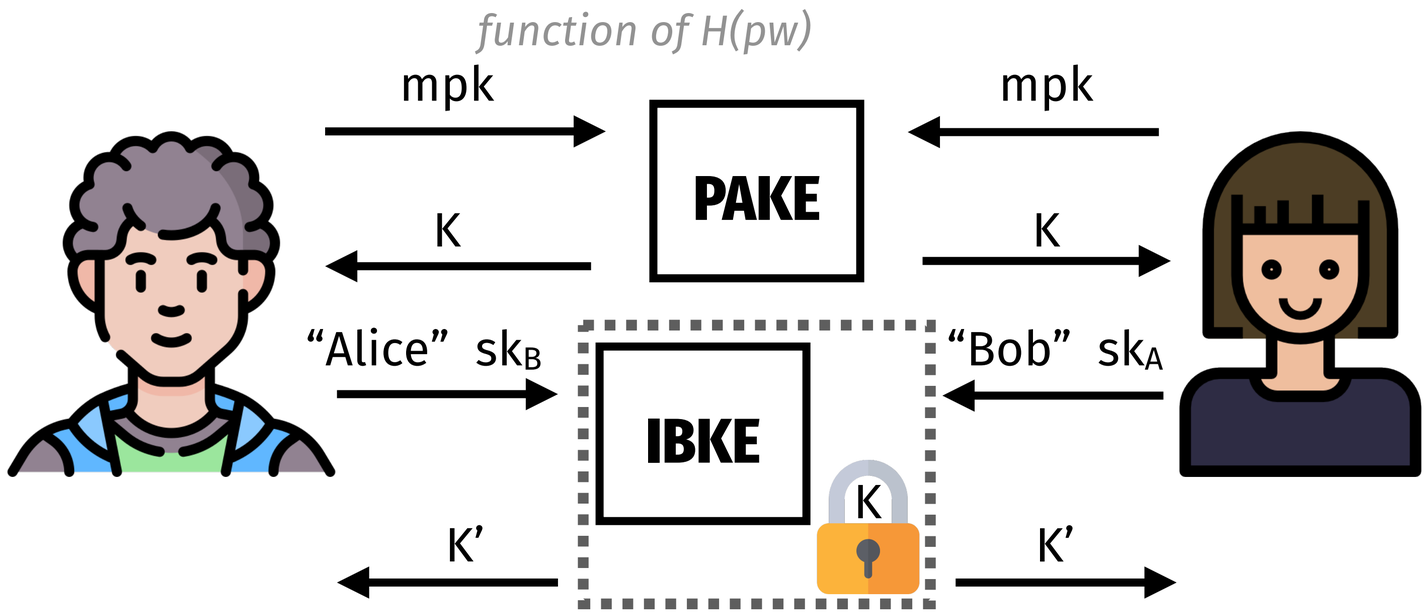
- "Secure channel" (EUE transform)
- How to make a PQ IBKE
- Security notion of IBKE (need KCIR+FS)

LATKE.Setup(pw, id) *same as CHIP*

$(msk, mpk) := KeyGen(1^\lambda; H(pw))$

$sk_{id} := Extract_{msk}(id)$

delete msk and pw



LATKE Construction

Use **self-KGC trick** of CHIP

Establish a secure channel like Ω method

Run **any IBKE in the secure channel**

Important details:

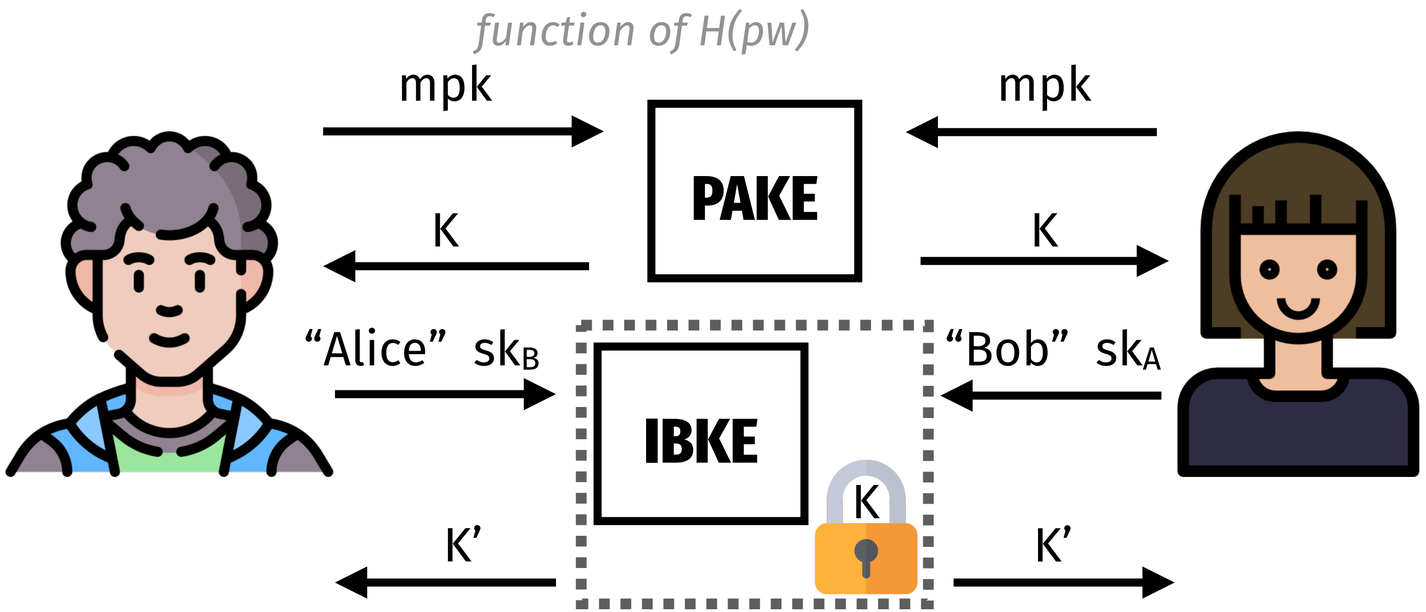
- "Secure channel" (EUE transform)
- How to make a PQ IBKE
- Security notion of IBKE (need KCIR+FS)

LATKE.Setup(pw, id) *same as CHIP*

(msk, mpk) := KeyGen(1^λ; H(pw))

sk_{id} := $Extract_{msk}$(id)

delete msk and pw



Bonus! Encrypted IBKE ⇒ identity concealment

LATKE Benchmarks

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

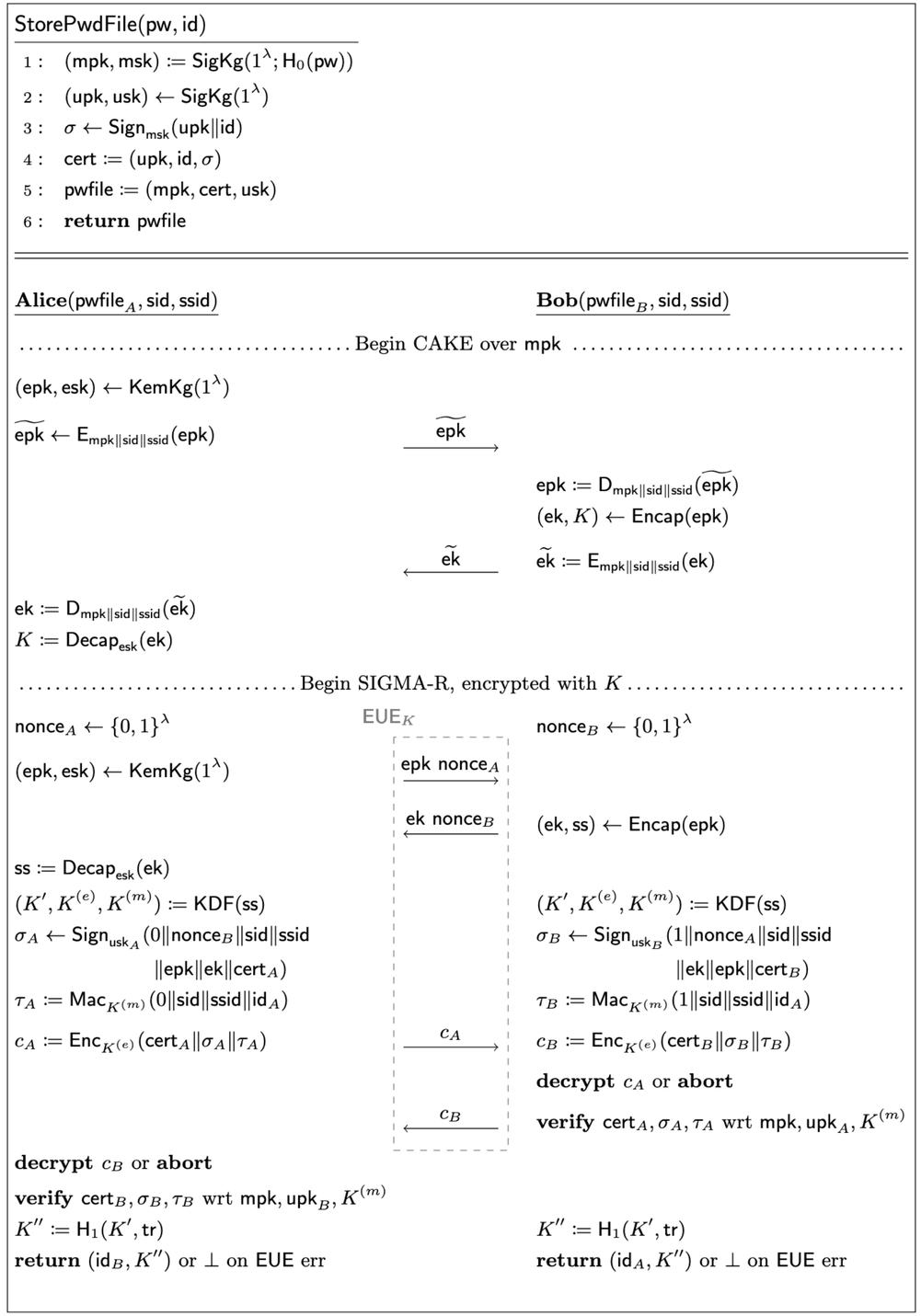
IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH,
(ID)SIGMA[Ed25519, Saber],
(ID)SIGMA[Dilithium, Saber]

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH, (ID)SIGMA[Ed25519, Saber], (ID)SIGMA[Dilithium, Saber]



LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH, (ID)SIGMA[Ed25519, Saber], (ID)SIGMA[Dilithium, Saber]

IoT believability: ran on my old home router

$\underline{\mathsf{StorePwdFile}(\mathsf{pw}, \mathsf{id})}$

$(\mathsf{mpk}, \mathsf{msk}) := \mathsf{SigKg}(1^\lambda; \mathsf{H}_0(\mathsf{pw}))$
$(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SigKg}(1^\lambda)$
$\sigma \leftarrow \mathsf{Sign}_{\mathsf{msk}}(\mathsf{upk} \| \mathsf{id})$
$\mathsf{cert} := (\mathsf{upk}, \mathsf{id}, \sigma)$
$\mathsf{pwfile} := (\mathsf{mpk}, \mathsf{cert}, \mathsf{usk})$
return pwfile

$\underline{\mathbf{Alice}(\mathsf{pwfile}_A, \mathsf{sid}, \mathsf{ssid})}$		$\underline{\mathbf{Bob}(\mathsf{pwfile}_B, \mathsf{sid}, \mathsf{ssid})}$
..... Begin CAKE over mpk		
$(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{KemKg}(1^\lambda)$		
$\widetilde{\mathsf{epk}} \leftarrow \mathsf{E}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\mathsf{epk})$	$\xrightarrow{\widetilde{\mathsf{epk}}}$	
		$\mathsf{epk} := \mathsf{D}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\widetilde{\mathsf{epk}})$
		$(\mathsf{ek}, K) \leftarrow \mathsf{Encap}(\mathsf{epk})$
	$\xleftarrow{\widetilde{\mathsf{ek}}}$	$\widetilde{\mathsf{ek}} := \mathsf{E}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\mathsf{ek})$
$\mathsf{ek} := \mathsf{D}_{\mathsf{mpk}\|\mathsf{sid}\|\mathsf{ssid}}(\widetilde{\mathsf{ek}})$		
$K := \mathsf{Decap}_{\mathsf{esk}}(\mathsf{ek})$		
..... Begin SIGMA-R, encrypted with K		
$\mathsf{nonce}_A \leftarrow \{0,1\}^\lambda$	EUE_K	$\mathsf{nonce}_B \leftarrow \{0,1\}^\lambda$
$(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{KemKg}(1^\lambda)$	$\xrightarrow{\mathsf{epk}\ \mathsf{nonce}_A}$	
	$\xleftarrow{\mathsf{ek}\ \mathsf{nonce}_B}$	$(\mathsf{ek}, \mathsf{ss}) \leftarrow \mathsf{Encap}(\mathsf{epk})$
$\mathsf{ss} := \mathsf{Decap}_{\mathsf{esk}}(\mathsf{ek})$		
$(K', K^{(e)}, K^{(m)}) := \mathsf{KDF}(\mathsf{ss})$		$(K', K^{(e)}, K^{(m)}) := \mathsf{KDF}(\mathsf{ss})$
$\sigma_A \leftarrow \mathsf{Sign}_{\mathsf{usk}_A}(0 \| \mathsf{nonce}_B \| \mathsf{sid} \| \mathsf{ssid} \| \mathsf{epk} \| \mathsf{ek} \| \mathsf{cert}_A)$		$\sigma_B \leftarrow \mathsf{Sign}_{\mathsf{usk}_B}(1 \| \mathsf{nonce}_A \| \mathsf{sid} \| \mathsf{ssid} \| \mathsf{ek} \| \mathsf{epk} \| \mathsf{cert}_B)$
$\tau_A := \mathsf{Mac}_{K^{(m)}}(0 \| \mathsf{sid} \| \mathsf{ssid} \| \mathsf{id}_A)$		$\tau_B := \mathsf{Mac}_{K^{(m)}}(1 \| \mathsf{sid} \| \mathsf{ssid} \| \mathsf{id}_A)$
$c_A := \mathsf{Enc}_{K^{(e)}}(\mathsf{cert}_A \| \sigma_A \| \tau_A)$	$\xrightarrow{c_A}$	$c_B := \mathsf{Enc}_{K^{(e)}}(\mathsf{cert}_B \| \sigma_B \| \tau_B)$
		decrypt c_A or **abort**
	$\xleftarrow{c_B}$	**verify** $\mathsf{cert}_A, \sigma_A, \tau_A$ wrt $\mathsf{mpk}, \mathsf{upk}_A, K^{(m)}$
decrypt c_B or **abort**		
verify $\mathsf{cert}_B, \sigma_B, \tau_B$ wrt $\mathsf{mpk}, \mathsf{upk}_B, K^{(m)}$		
$K'' := \mathsf{H}_1(K', \mathsf{tr})$		$K'' := \mathsf{H}_1(K', \mathsf{tr})$
return (id_B, K'') or $\perp$ on EUE err		**return** (id_A, K'') or $\perp$ on EUE err

LATKE Benchmarks

iPAKE	PQ?	Rounds	Comm.	Setup	Online	
Chip[Cpace,Fg] [CNPR22]	x	2	208B	284µs	5.33ms	(1×)
Latke[Cpace,FgC]	x	4	404B	314µs	5.56ms	(1.04×)
Latke[Cpace,IdHmqvC]	x	4	532B	467µs	5.62ms	(1.05×)
Latke[Cpace,IdSigDh]	x	2	616B	615µs	8.32ms	(1.56×)
Latke[Cake,IdΣEd25519]	enc.	6	3.53kB	813µs	5.46ms	(1.03×)
Latke[Cake,IdΣDilith2]	enc.+auth.	6	15.5kB	2.55ms	10.1ms	(1.89×)

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH, (ID)SIGMA[Ed25519, Saber], (ID)SIGMA[Dilithium, Saber]

IoT believability: ran on my old home router

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH, (ID)SIGMA[Ed25519, Saber], (ID)SIGMA[Dilithium, Saber]

IoT believability: ran on my old home router

iPAKE	PQ?	Rounds	Comm.	Setup	Online	
`Chip[Cpace,Fg]` [CNPR22]	x	2	208B	284µs	5.33ms	(1×)
`Latke[Cpace,FgC]`	x	4	404B	314µs	5.56ms	(1.04×)
`Latke[Cpace,IdHmqvC]`	x	4	532B	467µs	5.62ms	(1.05×)
`Latke[Cpace,IdSigDh]`	x	2	616B	615µs	8.32ms	(1.56×)
`Latke[Cake,IdΣEd25519]`	enc.	6	3.53kB	813µs	5.46ms	(1.03×)
`Latke[Cake,IdΣDilith2]`	enc.+auth.	6	15.5kB	2.55ms	10.1ms	(1.89×)

Pre-quantum LATKE is 4% slower than CHIP

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH, (ID)SIGMA[Ed25519, Saber], (ID)SIGMA[Dilithium, Saber]

IoT believability: ran on my old home router

iPAKE	PQ?	Rounds	Comm.	Setup	Online	
`Chip[Cpace,Fg]` [CNPR22]	x	2	208B	284µs	5.33ms	(1×)
`Latke[Cpace,FgC]`	x	4	404B	314µs	5.56ms	(1.04×)
`Latke[Cpace,IdHmqvC]`	x	4	532B	467µs	5.62ms	(1.05×)
`Latke[Cpace,IdSigDh]`	x	2	616B	615µs	8.32ms	(1.56×)
`Latke[Cake,IdΣEd25519]`	enc.	6	3.53kB	813µs	5.46ms	(1.03×)
`Latke[Cake,IdΣDilith2]`	enc.+auth.	6	15.5kB	2.55ms	10.1ms	(1.89×)

Pre-quantum LATKE is 4% slower than CHIP

Post-quantum is 3% slower

LATKE Benchmarks

Wrote LATKE and CHIP in Rust

PAKEs: CPace, CAKE[Saber]

IBKEs: (ID)HMQV, Fiore-Gennaro, (ID)Sig-DH, (ID)SIGMA[Ed25519, Saber], (ID)SIGMA[Dilithium, Saber]

IoT believability: ran on my old home router

iPAKE	PQ?	Rounds	Comm.	Setup	Online	
`Chip[Cpace,Fg]` [CNPR22]	x	2	208B	284µs	5.33ms	(1×)
`Latke[Cpace,FgC]`	x	4	404B	314µs	5.56ms	(1.04×)
`Latke[Cpace,IdHmqvC]`	x	4	532B	467µs	5.62ms	(1.05×)
`Latke[Cpace,IdSigDh]`	x	2	616B	615µs	8.32ms	(1.56×)
`Latke[Cake,IdΣEd25519]`	enc.	6	3.53kB	813µs	5.46ms	(1.03×)
`Latke[Cake,IdΣDilith2]`	enc.+auth.	6	15.5kB	2.55ms	10.1ms	(1.89×)

Pre-quantum LATKE is 4% slower than CHIP

Post-quantum is 3% slower

Tradeoff: rounds vs. speed

Future work

Future work

Strong iPAKE (siPAKE). An siPAKE is an iPAKE whose pwfiles cannot be precomputed. CRISP is (kinda) an siPAKE. Can we make a PQ one?

Future work

Strong iPAKE (siPAKE). An siPAKE is an iPAKE whose pwfiles cannot be precomputed. CRISP is (kinda) an siPAKE. Can we make a PQ one?

Round optimality. CHIP and LATKE are both at least 2 rounds. Can we do better?

Future work

Strong iPAKE (siPAKE). An siPAKE is an iPAKE whose pwfiles cannot be precomputed. CRISP is (kinda) an siPAKE. Can we make a PQ one?

Round optimality. CHIP and LATKE are both at least 2 rounds. Can we do better?

Hybrid iPAKE. LATKE is generic over PAKE and IBKE. Hybrid IBKE exists. Does a hybrid PAKE exist?

Future work

Strong iPAKE (siPAKE). An siPAKE is an iPAKE whose pwfiles cannot be precomputed. CRISP is (kinda) an siPAKE. Can we make a PQ one?

Round optimality. CHIP and LATKE are both at least 2 rounds. Can we do better?

Hybrid iPAKE. LATKE is generic over PAKE and IBKE. Hybrid IBKE exists. Does a hybrid PAKE exist?

Bonus points: why does the obvious hybrid PAKE not work?

Future work

Strong iPAKE (siPAKE). An siPAKE is an iPAKE whose pwfiles cannot be precomputed. CRISP is (kinda) an siPAKE. Can we make a PQ one?

Round optimality. CHIP and LATKE are both at least 2 rounds. Can we do better?

~~**Hybrid iPAKE.** LATKE is generic over PAKE and IBKE. Hybrid IBKE exists. Does a hybrid PAKE exist?~~

~~Bonus points: why does the obvious hybrid PAKE not work?~~

End notes

End notes

This paper is the result of >1yr of work

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

Fun tidbits not covered:

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

Fun tidbits not covered:

- LATKE is two iPAKEs: **post- and pre-specified peer model**

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

Fun tidbits not covered:

- LATKE is two iPAKEs: **post- and pre-specified peer model**
- Can make a **PAKE from just Saber + wide block cipher**. Saber is naturally an “obfuscated KEM” (see 2024/1086)

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

Fun tidbits not covered:

- LATKE is two iPAKEs: **post- and pre-specified peer model**
- Can make a **PAKE from just Saber + wide block cipher**. Saber is naturally an “obfuscated KEM” (see 2024/1086)
- Needed to (slightly) **modify ideal functionality $\mathbf{F_{iPAKE}}$**

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

Fun tidbits not covered:

- LATKE is two iPAKEs: **post- and pre-specified peer model**
- Can make a **PAKE from just Saber + wide block cipher**. Saber is naturally an “obfuscated KEM” (see 2024/1086)
- Needed to (slightly) **modify ideal functionality F_{iPAKE}**
- **Fixed CHIP’s proof of security**

End notes

This paper is the result of >1yr of work

- The **first version was so broken** that it had to be retracted

Fun tidbits not covered:

- LATKE is two iPAKEs: **post- and pre-specified peer model**
- Can make a **PAKE from just Saber + wide block cipher**. Saber is naturally an “obfuscated KEM” (see 2024/1086)
- Needed to (slightly) **modify ideal functionality F_{iPAKE}**
- **Fixed CHIP’s proof of security**

Read the paper if you find this cool!

Conclusion

Constructed **LATKE** an identity-binding PAKE

Generic: takes any PAKE and (nearly) any IBKE. Hence, **PQ**

Fast: As low as 3% overhead compared to CHIP

ia.cr/2023/324
github.com/rozbb/latke-ipake
research@mrosenberg.pub

Icon credits: Flat Icons, SmashIcons, Assia Benkerroum, Freepik, Those Icons